Angerhausen · Brückmann
Englisch

VC-20 intern

Betriebssystem und Technik des VC-20

EIN DATA BECKER BUCH

Angerhausen · Brückmann
Englisch

VC-20 intern

Betriebssystem und Technik des VC-20

EIN DATA BECKER BUCH

VORWORT

Wer tiefer in Geheimnisse und Fähigkeiten des Commodore VC-20 eindringen möchte, muß sich mit Maschinensprache befassen. Wir gaben zur Unterstützung der ernsthaften Maschinensprache-Programmierer im vorigen Jahr ein ROM-Listing des VC-20 heraus, das schnell vergriffen war.

Die folgende 2. Auflage bekam den Namen VC-20 intern. Sie wurde erheblich erweitert und überarbeitet. Wir hatten eine Einführung in die Maschinensprache hinzugefügt, da sich mehr und mehr VC-20 Besitzer für dieses Thema interessierten.

Die nun vorliegende 3. Auflage wurde wiederum erweitert, da uns das Echo aus dem Leserkreis zeigte, daß zur intimen Kenntnis und effektiven Nutzung des VC-20 gerade unter Gebrauch von Maschinenspracheprogrammen der Einstieg in die Hardware des Gerätes unumgänglich ist.
Sie finden daher in dieser Auflage sowohl den vollständigen Schaltplan des VC-20, wofür wir an dieser Stelle der Commodore Büromaschinen GmbH für die Erlaubnis zum Abdruck danken möchten, als auch eine ausführliche Beschreibung desselben, sowie Registerbeschreibungen des VIC-Chip und der VIA.

Autoren sind:

Michael Angerhausen
und Lothar Englisch

Für Schaltplan- und Registerbeschreibung sorgten:

Rolf Brückmann
und Klaus Gerits

Manuskript, Abwicklung und Gestaltung besorgten:

Alicia Clees
und Cäcilia Kalkowski

Allen sechsen möchte ich hiermit herzlich danken und wünsche Ihnen eine angenehme Lektüre.

Düsseldorf, im September 1983

- Dr. Achim Becker -

Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.
Alle Schaltungen, technische Angaben und Programme in diesem Buch wurden von den Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler sind die Autoren jederzeit dankbar.

MASCHINENPROGRAMMIERUNG AUF DEM VC-20

*** EINFÜHRUNG ***

Es gibt viele neue Programmiersprachen für Computer - warum beschäftigt man sich da noch mit einer Sprache, die Ihre Wurzeln in den ersten Tagen der "Computerei" hat ?

Die Sprache des VC-20 ist BASIC, das ist allen bekannt. Aber wie kann der Computer die Befehle, die Sie ihm geben, verstehen und eine bestimmte Aufgabe lösen ? Dazu gehört schon etwas mehr als BASIC. Um das zu verstehen, muß man sich mit den Grundlagen der Programmierung befassen. Das wollen wir in diesem Kapitel nun machen.

Die Programmiersprache BASIC wird vom Computer eigentlich überhaupt nicht verstanden. Der Programmierer, also Sie, können zwar in BASIC programmieren, der Computer muß jeden Befehl aber erst in eine für ihn verständliche "Sprache" übersetzen (interpretieren). Aus diesem Grund nennt man die BASIC Version des VC-20 auch BASIC INTERPRETER. Jeder Befehl wird also auf eine ganz bestimmte Art und Weise dekodiert und so die Informationen für den Computer aufbereitet. Die Sprache in der dies geschieht nennt man MASCHINEN-SPRACHE. Nun wird schon deutlich warum man diese Sprache einsetzt:

Zur Programmierung von Betriebssystemen und deren Erweiterungen und Hilfen und vor allen Dingen für Programme die sehr schnell sein müssen (z.B. für die Steuerung von Anlagen von einem Computer aus). Auch wenn die Maschinensprache schon etwas alt ist: Jeder professionelle Programmierer sollte sich in der Maschinensprache auskennen.

Warum programmiert man dann nicht einfach gleich in Maschinensprache ? Ganz einfach. Die Programmierung in Maschinensprache ist bei weitem nicht so komfortabel wie die Programmierung in BASIC. Sie ist bedeutend zeitaufwendiger und schwieriger.

Jeder BASIC-Befehl entspricht einer ganzen Reihe von Maschinenbefehlen. Ein kleines Beispiel:

Eine Addition zweier Zahlen

Das Ergebnis muß bei dem Programm in Maschinensprache kleiner oder gleich 255 sein, da man mit dieser sogenannten 8-Bit Operation nur Zahlen zwischen 0 und 255 bzw. zwischen -128 und +127 darstellen kann. Für größere Zahlen muß man auf die 16-Bit Addition zurückgreifen. In BASIC würde die Programmzeile so aussehen:

```
PRINT 6 + 5
```

Die ganze Operation läßt sich also leicht in einer Zeile darstellen. In Maschinensprache sähe das Resultat so aus:

```
CLC              ; LöSCHE DEN ÜBERTRAG
LDA  #$06        ; LADE AKKU MIT 6
ADC  #$05        ; ADDIERE AKKU + ÜBERTRAG + 5
JSR  AUSGABE     ; DRUCKE ERGEBNIS (=11 oder B)
```

```
                .     .        ;
                .     .        ; HIER KöNNEN WEITERE BEFEHLE STEHEN
                .     .        ;
AUSGABE        PHA             ; RETTE DEN AKKUMULATOR
               LSR   A         ; BRINGE B7 BIS B4 AND DIE STELLE
               LSR   A         ;  VON B3 BIS B0 - DIE FREIEN
               LSR   A         ;  STELLEN (B7 BIS B4) WERDEN DANN
               LSR   A         ;  ZU 0
               JSR   HEXASC    ; WANDLE DIE ZAHL IN EIN ZEICHEN
               PLA             ; HOLE AKKU VOM STAPEL
               AND   #$0F      ; LöSCHE B7 BIS B4
HEXASC         CLC             ; LöSCHE üBERTRAG
               ADC   #$F6      ; ADDIERE F6 (=246)
               BCC   $04       ; ERGEBNIS <= 255 ? JA ! 4 BYTES WEITER
               ADC   #$06      ; ADDIERE 6
               ADC   #$3A      ; ADDIERE 3A (=58)
               JMP   $FFD2     ; DRUCKE ZEICHEN AUS AKKU
                .     .        ;
                .     .        ; HIER KANN DAS PROGRAMM WEITERGEHEN
                .     .        ;
```

Diese Routinen können Sie für Ihre Programme verwenden. Die Bedeutung dieser Routinen werden später noch erläutert.

*** WIE VERSTEHT DER COMPUTER UNSERE BEFEHLE ? ***

Wie Sie sicherlich wissen, kennt man in der Elektronik zwei elektrische Zustände: STROM AN oder STROM NICHT AN.

Diese Zustände finden wir auch in der Computertechnik. Nicht nur in der HARDWARE, sondern auch in der SOFTWARE und vor allen Dingen in der Maschinenprogrammierung, spielen diese Zustände eine wesentliche Rolle. Diese beiden Zustände bilden nämlich die kleinstmögliche INFORMATIONSEINHEIT die wir kennen: ein BIT.

Aus dieser Idee heraus findet hier das System der DUALZAHLEN seine Anwendung. Dieses Zahlensystem umfaßt nur zwei Zahlen (daher DUAL). Diese Zahlen lesen wir als 0 und 1. Sie haben für uns eine ganz bestimmte Bedeutung:

0 - STROM NICHT AN
1 - STROM AN

Aus der Kombination von diesen beiden Zahlen lassen sich alle möglichen Zahlen darstellen. Bei einem 8-Bit Prozessor (z.B. dem 6502 des VC-20) sieht die Darstellung folgendermaßen aus:

b7 b6 b5 b4 b3 b2 b1 b0

Dabei steht jedes kleine 'b' für BIT. Nun hat jede Stelle ihren festen Wert. Dieser Wert läßt sich leicht über die entsprechende Zweierpotenz errechnen:

```
b7 = 2**7 = 128
b6 = 2**6 =  64
b5 = 2**5 =  32
b4 = 2**4 =  16
b3 = 2**3 =   8
```

b2 = 2**2 = 4
b1 = 2**1 = 2
b0 = 2**0 = 1

(2**x wird gesprochen als "2 hoch x")

Die Zahl 15 würde demanch dargestellt als:

0 0 0 0 1 1 1 1

und die Zahl 16 als:

0 0 0 1 0 0 0 0

Die größte darstellbare Zahl ist also 255, oder:

1 1 1 1 1 1 1 1

Die Bedeutung dieser Zahlen kann man dann Zahl für Zahl aus der oben angegebenen Tabelle entnehmen.

Um die Maschinensprache richtig zu verstehen, und anzuwenden, ist es notwendig auch das Prinzip der Dualzahlen zu verstehen. Denn auf dieser Ebene arbeitet der Computer. Wenn Sie mit diesem Zahlensystem noch nicht so vertraut sind, empfehlen wir Ihnen die Lektüre eines der vielen 6502 Handbüchern (z.B. PROGRAMMIERUNG DES 6502 VON RODNEY ZAKS - SYBEX VERLAG).

Nachdem wir nun einen kurzen Einblick in das Dualzahlen-System getan haben, wollen wir jetzt auf die eigentliche Maschinenprogrammierung kommen.

Bei dem Vergleich von BASIC und Maschinensprache wurde deutlich, daß man bei der Verwendung der maschinenorientierten Sprache in der Programmierung etwas anders vorgehen muß als bei der normalen BASIC-Programmierung. Es ist für den Anfänger sicherlich nicht ganz einfach, diese neue Art der Programmierung zu verstehen. Wenn man aber seinen Computer beherrschen will - und wer will schon, daß es umgekehrt ist - sollte man sich mit der Maschinensprache beschäftigen.

Denkt man einmal daran eine schnelle Sortier- oder Suchroutine zu schreiben (z.B für eine Dateiverwaltung), so kommt man einfach nicht an der Maschinensprache vorbei. Und für denjenigen der sein Betriebssystem richtig verstehen will, den gibt es überhaupt keine andere Möglichkeit als die Maschinensprache.

Sie haben in diesem Kapitel vielleicht festgestellt, daß es Ihnen noch schwer fällt sich mit der Maschinensprache zu befassen. Sie sollten sich dann nicht scheuen die entsprechenden Kapitel mehrmals durchzulesen. Denn auch in der Programmierung gilt die Regel:

NUR ÜBUNG MACHT DEN MEISTER !

*** DER MONITOR - UND WAS STECKT DAHINTER ***

Ein sehr nützliches, ja beinahe unentbehrliches Werkzeug zur Maschinenprogrammierung, ist der sogenannte MONITOR. Der Monitor ist ein reines Hilfsmittel (Er hat nichts zu tun mit einem Datensichtgerät !). Er ermöglicht das Verändern von Speicherplätzen und Registern, die Ausführung von Maschinenprogrammen und das Laden, Abspeichern und Disassemblieren von Maschinenprogrammen. Schrecken Sie jetzt nicht vor diesem "Fachchinesisch" zurück. Wir gehen später genau auf jeden Begriff ein.

Ein weiterer Vorzug des Monitors ist die Möglichkeit mit ihm leicht und problemlos Maschinenprogramme zu erstellen und zu ändern.

Wir wollen nun einmal zeigen wie man mit einem Standard-Monitor arbeitet.

Nachdem der Monitor von Kassette oder Diskette geladen wurde, startet man ihn entweder mit dem normalen RUN oder aber mit einem Aufruf für ein Maschinenprogramm: SYS xxxx. Die 'xxxx' stehen hier natürlich für die Startadresse des Monitors. Diese Adressen sind von Monitor zu Monitor verschieden.

Als nächstes meldet sich der Monitor mit einem PROMPT ('Hier bin ich'). Er erwartet nun von Ihnen die Eingabe von irgendwelchen Befehlen. Die Meldung nach dem Aufruf des Monitors könnte zum Beispiel so aussehen:

```
C*
     PC   IRQ  SR AC XR YR SP
>;  E142  EAA1 31 32 AC 34 F8
>
```

Dieses zeigt dem Programmierer die momentanen Inhalte der verschiedenen REGISTER. Beim 6502 kennen wir folgende Register:

- PC - Der Programmzähler. In diesem Register steht immer die nächste zu verarbeitende Programmadresse.
- IRQ - Der Interruptvektor. Dieses Register steuert eine mögliche Programmunterbrechung (INTERRUPT).
- SR - Das Statusregister. In diesem Register werden die einzelnen Zustände der FLAGGEN (Wegweiser) des Prozessors verwaltet.
- AC - Der Akkumulator. In diesem Register werden alle Vergleichs- und Rechenoperationen ausgeführt.
- XR - Das X-Regsiter. Das Register dient zur Aufnahme von bestimmten Daten während des Programmablaufs.
- YR - Das Y-Register. Auch dieses Register dient zur Aufnahme von Daten während des Programmablaufs.
- SP - Der Stapelzeiger. Dieses Register zeigt auf einen bestimmten Wert im Stapel (STACK).

Der Inhalt dieser Register kann vom Monitor laufend überprüft und verändert werden. Zu beachten ist, daß alle Eingaben von Zahlen in hexa-dezimaler Form zu erfolgen haben (sofern dies nicht ausdrücklich anders angegeben ist). Wenn Sie also die

Zahl 255 eingeben wollen müssen Sie die Hex-Zahl FF eintragen. Diese Eintragung geschieht durch einfaches überschreiben des alten Wertes. Sie fahren also mit dem Cursor durch Drücken der Pfeil-Tasten über den entsprechenden Wert und geben dann den neuen Wert ein. Nach Betätigung der RETURN-Taste übernimmt der Computer den neuen Registerinhalt.

Genauso wie alle Register, so lassen sich natürlich auch die Speicherinhalte anzeigen und verändern. Dies ermöglicht dann die Eingabe von Maschinenprogrammen. Der Befehl um einen bestimmten Speicherplatz anzuzeigen lautet:

M xxxx yyyy

wobei M von der Tastatur aus eingegeben wird und xxxx/yyyy die entsprechenden Anfangs und Endadresse des auszugebenden Speicherbereichs angeben.

Nach dieser Eingabe werden dann die entsprechenden Speicherinhalte angezeigt. Wenn der angegebene Speicherbereich nicht vollständig auf eine Bildschirmseite geht, verschiebt sich der Text Zeile für Zeile nach oben. Durch Drücken von RUN/STOP kann man wieder in den Kommando-Modus zurückkehren.

Genau wie beim Ändern der Registerinhalte, so werden auch hier die einzelnen Speicherstellen verändert. Das heißt, daß man mit dem Cursor an die entsprechende Stelle auf dem Bildschirm fährt, dort den neuen Wert oder die neuen Werte eingibt und danach die RETURN-Taste drückt.

So könnte zum Beispiel eine Ausgabe von einem bestimmten Speicherbereich aussehen:

```
>M C000 C010
>: C000  A9 10 8D 16 03 A9 C0 8D
>: C008  17 03 A9 43 85 97 D0 16
>: C010  A9 42 85 97 D8 4A 68 8D
```

Wenn Sie diese Ausgabe 'lesen' wollen, müssen Sie so vorgehen: Sie nehmen die Adresse, welche am Anfang jeder Zeile steht (z.B. C000 in der ersten Zeile) und addieren dann jeweils die Position des Wertes. In unserem Beispiel hieße das: An der Speicherstelle C000 steht der Wert A9, an der Stelle C001 der Wert 10 usw. Denken Sie aber immer daran, daß <u>alle</u> Angaben, also auch die Angaben der Adressen in hexadezimal erfolgen. Nach der Adresse C009 folgen also die Adressen C00A, C00B, C00C, C00D, C00E und C00F. Erst dann folgt die Adresse C010.

Der Speicher kann aber noch auf eine andere Art und Weise ausgegeben werden. Durch das DISASSEMBLIEREN von dem jeweiligen Speicherbereich.

Das Disassemblieren hat den Vorteil, daß aus den unübersichtlichen Hex-Zahlen leicht zu verstehende Befehlsfolgen werden. Diese Befehlsfolgen nennt man MNEMONICS. Die mehr oder weniger aufwendige "Übersetzung" der Hex-Zahlen in "Klartext" für den Programmierer entfällt beim Disassemblieren. Allerdings hat das Disassemblieren auch

einen Nachteil:

Wenn irgendwo im Maschinenprogramm zum Beispiel ein Text, der nichts mit den eigentlichen Befehlen zu tun hat, erscheint, versucht der Disassembler diesen Text als OPERATIONSCODES (kurz OPCODES) zu interpretieren. Es kann vorkommen, daß manche OPCODES den selben Wert wie normale Buchstaben haben (Sie kennen ja den BASIC-Befehl CHR$ (x) ?). Dann erscheint die entsprechende (falsche !!!) Mnemonics für den Wert. Es kann aber auch vorkommen, daß der Disassembler den Wert überhaupt nicht versteht. Dann erscheinen bei der Ausgabe an der jeweiligen Stelle Fragezeichen '???'. Ein erfahrener Programmierer wird aber schnell sehen was 'richtige' und was 'falsche' Werte sind.

Der nächste Schritt nach der Eingabe oder Änderung eines Maschinenprogramms ist dann natürlich die PROGRAMMAUSFÜHRUNG. Dies geschieht mit dem Befehl

G xxxx

G bedeutet dabei soviel wie GO TO Speicherstelle. Dieser Befehl bewirk also einen Sprung zu einer bestimmten Adresse (xxxx) und führt dann das dort stehende Maschinenprogramm aus. Trifft das Maschinenprogramm auf einen BRK (BREAK) Befehl, so wird die Programmausführung abgebrochen und der Monitor meldet sich mit

B*

und befindet sich wieder im Kommando-Modus. Dazu werden noch die Inhalte aller Register angezeigt. Der Programmzähler hat nun die nächste Adresse nach dem Abbruch gespeichert. Durch geschicktes Einfügen dieses Befehls innerhalb eines Maschinenprogrammes ist es dem Programmierer sehr leicht möglich die Maschinenprogramme zu testen ohne direkt einen 'Absturz' befürchten zu müssen. Nach diesem Austesten des Programms kann man dann den BRK-Befehl wieder entfernen und gegen den eigentlichen Befehl dieser Adresse austauschen.

Ist Ihnen aufgefallen, daß auch Maschinenprogrammbefehle, wie z.B. BRK, für den Neuling doch recht verständlich werden ? Im übrigen hat dieser Befehl natürlich die selbe Wirkung wie der STOP-Befehl in BASIC.

Natürlich sollen Maschinenprogramme nicht nur geschrieben, geändert und getestet, sondern auch gespeichert und wieder geladen werden. Zum SPEICHERN von Maschinenprogrammen auf Band oder Diskette gibt es den Befehl

S "NAME",xx,yyyy,zzzz

Hierbei bedeutet "NAME" natürlich eine beliebige Bezeichnung für das Programm. Das Kürzel 'xx' gibt die Geräteadresse an auf der das Programm gespeichert werden soll. Als Werte sind bei Commodore möglich:

01 - Kassette. Dieser Wert muß nicht angegeben werden.
08 - Diskette. Bei Ausgabe auf Disk muß dieser Wert mit angegeben werden.

Die nächsten zwei Werte geben die Start- bzw. Endadresse (+1) des zu speichernden Speicherbereichs an. Wollen Sie also zum Beispiel ein Programm das bei $0800 beginnt ($ bedeutet Hex) und bei $0FFF endet, auf Diskette abspeichern, so sähe der Befehl dafür so aus:

S "NAME",08,0800,1000

Nach dem Drücken der RETURN-Taste wird das Maschinenprogramm dann auf die Diskette abgespeichert.

Das LADEN eines Programmes von Kassette / Diskette erfolgt ähnlich dem Schreiben. Allerdings brauchen hier die Anfangs- und Endadressen nicht mit angegeben zu werden, da der Computer aus dem "Vorspann" des Programmes selbst die nötigen Adressen feststellt. Es gibt allerdings auch Monitore bei denen es möglich ist die Adressen anzugeben und so das Programm an beliebige Adressen zu legen (RELOKATIEREN). Es ist dabei aber zu beachten, daß nicht alle Befehle relokatiert werden. So bleibt zum Beispiel ein absoluter Sprung nach $0900 ein Sprung nach $0900 auch wenn dort das Programm überhaupt nicht liegt. Nur indirekte Sprünge können korrekt ausgeführt (relokatiert) werden. Der allgemeine Befehl zum Laden von Programmen lautet:

L "NAME",xx

wobei 'xx' wieder die Angabe über das Gerät, von der das Programm geladen werden soll, enthält.

Ein weiterer interessante Befehl ist der X (EXIT, Abbruch) Befehl. Dieser Befehl erlaubt die Rückkehr ins BASIC ohne ein bestehendes Maschinenprogramm zu verändern. Sie können also mit dem Monitor ein Maschinenprogramm erstellen und dann ins BASIC zurückkehren, um dort beispielsweise ein BASIC Programm zu erstellen. Danach haben Sie wieder die Möglichkeit in den Monitor zu gehen usw.

Ein Tip am Rande: Wenn Sie ein Maschinenprogramm geladen haben (z.B. einen Monitor) empfiehlt es sich im Direkt-Modus, also direkt über die Tastatur, den NEW-Befehl einzugeben. Dadurch werden alle wichtigen Adressen wieder richtig gesetzt. Ansonsten könnte es passieren, daß Sie bei Eingabe eines BASIC-Programmes schon nach der ersten Zeile eine OUT OF MEMORY Fehlermeldung bekommen (oder andere recht eigenartige Sachen passieren).

Natürlich kann man auch ohne Monitor in Maschinensprache programmieren und diese starten. Doch muß man dann beachten, daß jeder einzelne Speicherplatz von BASIC aus geändert werden muß.

Das bedeutet beispielsweise, daß Sie ein entsprechendes BASIC Programm schreiben müßten, mit dem Sie dann die Operationscodes eingeben können. Dies kann aber mit Schwierigkeiten verbunden sein. Fast jedes Maschinenprogramm-Listing ist in hexa-dezimaler Form angegeben. Da der POKE-Befehl in BASIC (durch den werden ja einzelne Speicherplätze verändert) aber nur dezimale Zahlen erlaubt, müßten Sie alle Hex-Werte in dezimale Werte umrechnen. Erst dann kann der

POKE-Befehl angewendet werden. Bei der Ausgabe von Speicherplätzen muß dann genau in der anderen Reihenfolge vorgegangen werden. Zuerst also den Wert mit PEEK lesen, ausgeben und dann in Hex-Werte umsetzen.

Ein anderes Problem sind die Speicherplätze, die von BASIC belegt werden. Um das System vor einem Absturz zu bewahren, muß darauf geachtet werden, keine Adressen zu verändern, oder Speicherbereiche zu verwenden, die auch von BASIC (bzw. dem BASIC Programm) belegt werden.

Probleme gibt es außerdem bei der Abspeicherung von Maschinenprogrammen, da im normalen BASIC-Modus nicht die Möglichkeit besteht, bestimmte Speicherbereiche abzuspeichern. BASIC nimmt bei dem SAVE-Befehl einfach die Angaben die in den entsprechenden Registern (Anfang BASIC Programm - Ende BASIC Programm) stehen und fügt diese automatisch an den SAVE-Befehl an. Sie würden also versuchen ein ganz normales BASIC Programm abzuspeichern.

Wie Sie sehen, ist die Anschaffung eines in der Regel recht preisgünstigen Monitors jedem angehenden Systemprogrammierer (oder denen die es werden wollen) unbedingt zu empfehlen.

*** WIE GEHE ICH BEI DER MASCHINENPROGRAMMIERUNG VOR ? ***

Wenn Sie nun mit dem Monitor und seiner Anwendung einigermaßen vertraut sind, kann mit der eigentlichen Programmierung begonnen werden.

Der gesamte Ablauf des Programmierens läßt sich in 5 Schritte aufteilen:

1.) Der erste Schritt ist die Erstellung eines Programmablaufplanes (PAP). In der professionellen EDV bestehen diese PAP aus genormten Symbolen. Für uns reicht es, wenn wir uns einen sprachlich kurz formulierten Stichpunktzettel machen. Auf diesem Zettel steht dann der grob entworfene Ablauf des Programmes (Verzweigungen, benötigte Peripherie etc.). An Hand dieses Blattes können wir in jedem Stadium unserer Programmierung feststellen, was zu tun ist.

Wer will, kann natürlich auch professionelle PAP erstellen. Es gibt sehr viele Bücher, die die Erstellung dieser Ablaufpläne beschreiben. Dann werden auch die Programmier von Groß-Rechnern feststellen, daß unsere Personalcomputer kein Spielzeug sind, auf denen man nichts vernünftiges machen kann.

Außerdem haben Sie so nach kurzer Zeit eine große Sammlung sogenannter Algorithmen beisammen, die Sie in alle Programmiersprachen übersetzen können.

Dieser erste Schritt wird oft vernachlässigt (LEIDER !!!!), aber gerade dieser Schritt, entscheidet oft über die Qualität eines Programmes. Das endgültige Programm sollte also schon in dieser Phase auf dem Papier existieren.

2.) Eine sehr wichtige Überlegung bei der Programmierung in Maschinensprache (und auch später bei der Assembler-Programmierung) ist die Wahl des Speicherbereiches in dem das Programm ablaufen soll. Maschinenprogramme arbeiten sehr oft mit Programmen höherer Programmiersprachen (z.B. BASIC, PASCAL, COBOL, FORTRAN etc.) zusammen. Da diese Sprachen genau wie das verwendete Betriebssystem, einen bestimmten Speicherplatz benötigen, um dort Daten, Zeiger und dergleichen zu speichern, muß das Maschinenprogramm vor einer Überschreibung und damit Zerstörung geschützt werden. Dies kann durch einen kleinen Trick geschehen: Man muß dem Computer mitteilen, daß ihm ein Teil des Speichers nicht mehr zur Verfügung steht. Das Programm kann dann weiterhin auf das Maschinenprogramm zugreifen, ohne aber dieses zu überschreiben.

Eine andere Möglichkeit der Platzierung von Maschinenprogrammen besteht darin, das Maschinenprogramm in einen Speicherbereich zu legen, der von der Programmiersprache und dem Betriebssystem nicht beeinfluß wird.

3.) Erst wenn die beiden vorherigen Schritte sorgfältig durchgeführt sind, sollte mit der eigentlichen Programmierung

begonnen werden. Diese ganzen Schritte sind gleichermaßen für Maschinen- als auch für die sonstige Programmierung sinngemäß die Selben.

Die Frage, die man sich nun stellt, betrifft die Art und Weise, das endgültige Programm zu erstellen. Entweder schreibt man das Programm nur mit den jeweiligen Opcodes (also 'zu Fuß'), oder man verwendet, sofern man den entsprechenden Monitor dafür besitzt, die Mnemonics. Diese Art kommt der professionellen Assemblerprogrammierung schon etwas näher. Aber darauf kommen wir später noch.

4.) Der vierte Punkt lehnt sich stark an den dritten Punkt an. Hier muß man entscheiden, wie man das nun fertige Maschinenprogramm in den Speicher bringt. Die einfachste Möglichkeit ist, durch den BASIC-Befehl POKE jede Adresse einzeln zu verändern. Das ist natürlich sehr umständlich und mühsam. Außerdem hat man keine genaue Kontrolle über die eingegebenen Opcodes. Deshalb ist auch hier wieder ein Monitor die beste Möglichkeit.

Um zu testen, ob die eingegebenen Opcodes auch korrekt waren, brauchen Sie nur, nachdem Sie die ganzen Opcodes eingegeben haben, mit dem Disassemblier-Befehl

D xxxx yyyy

den entsprechenden Speicherbereich (xxxx = Startadresse, yyyy = Endadresse) zu disassemblieren und mit Ihren Notizen auf dem Papier zu vergleichen. Wenn alles korrekt war kommt der nächste Schritt.

5.) Als letzter Punkt bleibt noch der Testlauf. Hier stellt sich dann heraus, ob alles so läuft wie man sich das vorgestellt hat. Hier rächt sich dann auch eine nicht ausreichende Vorbereitung (s. Punkt 1). Wenn irgendwo im Programm ein Fehler auftreten sollte - und das passiert auch den besten Programmieren - kann man durch Einsetzen des BRK-Befehls im Programm mit Hilfe des Monitors leicht den Fehler finden.

Beim nächsten mal klappt es dann bestimmt.

Wenn Sie diese 5 Punkte zu Ihrem Leitfaden machen, werden Sie beim Programmieren keine bösen Überraschungen erleben. Es ist schon oft vorgekommen, daß jemand sein eigenes Programm nach ein paar Wochen selber nicht mehr verstand, nur weil die richtige Vorbereitung und die notwendigen Aufzeichnungen fehlten. Gewöhnen Sie sich daher von vorne herein einen korrekten Programmierstil an.

Wir wollen Ihnen einmal an einem kleinen Beispiel den Ablauf einer korrekten Programmentwicklung geben. Das Programm soll einen kleinen Text auf dem Bildschirm ausgeben. Was benötigen wir dafür ?

Als erstes wollen wir den kleinen Text von der Tastatur aus einlesen und zwar bis die RETURN-Taste gedrückt wird. Danach soll der gleiche Text noch einmal auf dem Bildschirm ausgegeben werden. Für dieses kleine Problem befinden sich schon einige nützliche Routinen im ROM des VC 20, auf die Sie zurückgreifen können. (Vergleichen Sie auch die Tabelle mit den Routinen in diesem Buch)

Als nächstes müssen wir einen passenden Speicherplatz für unser kleines Programm finden. Es besteht die Möglichkeit kleinere Maschinenprogramme, so wie dieses Beispielsprogramm in den Kassettenpuffer zu legen. Dieses hat aber einen Nachteil: Wir können das Programm nicht auf Kassette abspeichern, da dieser Kassettenpuffer beim Speichern mit eigenen Daten (z.B. der Name des Programmes) überschrieben wird - das Programm wäre also zerstört. Da dies allerdings nur ein kleines Übungsprogramm werden soll, können wir es ruhig an diese Stelle speichern. Das hat den Vorteil, daß dieses Programm auf jedem VC 20 läuft - die Erweiterung wird dabei ja nicht genutzt.

Unser kleines Programm soll also ab Adresse $033C (Beginn des Kassettenpuffers) beginnen. Das Programm würde in Assembler geschrieben so aussehen:

Adresse	Wert	Opcode	Operand	Bemerkung
033C	20 5F E5	JSR	$E55F	; LÖSCHE BILDSCHIRM
033F	A2 00	LDX	#$00	; LÄNGE DES TEXTES=0
0341	20 CF FF	JSR	$FFCF	; TASTATUR LESEN
0344	C9 00	CMP	#$00	; KEINE TASTE GEDR.?
0346	F0 F9	BEQ	$0341	; NEIN - ZURÜCK !
0348	C9 0D	CMP	#$0D	; RETURN-TASTE ?
034A	F0 07	BEQ	$0353	; JA - AUSGABE !
034C	9D 00 02	STA	$0200,X	; SPEICHER ZEICHEN !
034F	E8	INX		; LÄNGE TEXT+1
0350	4C 41 03	JMP	$0341	; WIEDER LESEN
0353	A9 00	LDA	#$00	; AKKU=0
0355	9D 00 02	STA	$0200,X	; SPEICHER ZEICHEN !
0358	A9 00	LDA	#$00	; LSB
035A	A0 02	LDY	#$02	; MSB
035C	20 1E CB	JSR	$CB1E	; DRUCKE TEXT !
035F	4C 74 C4	JMP	$C474	; ZURÜCK INS BASIC !

Der Speicherauszug mit dem Monitor sieht folgendermaßen aus:

```
>M 033C 0361
>: 033C 20 5F E5 A2 00 20 CF FF
>: 0344 C9 00 F0 F9 C9 0D F0 07
>: 034C 9D 00 02 E8 4C 41 03 A9
>: 0354 00 9D 00 02 A9 00 A0 02
>: 035C 20 1E CB 4C 74 C4 .. ..
```

Unser kleines Programm ist insgesamt 25 Bytes lang. Es zeigt

zwar nur ein paar Befehle der sehr umfangreichen Befehlsliste des 6502, aber es wird nun hoffentlich doch das Prinzip sichtbar, das hinter der Maschinenprogrammierung steckt.

Man beginnt stets mit der INITIALISIERUNG der Variablen (d.h. sie auf einen Anfangs- oder Ausgangswert zu bringen). Außerdem wird noch die Periherie (z.B. Bildschirm oder Drucker) mit eingebunden (dem Gerät wird mitgeteilt welche Zusatzgeräte noch angeschlossen sind). Wenn das geschehen ist, kann das eigentliche Programm gestartet werden. Wir wissen nun schon, daß dafür der SYS-Befehl verwendet wird:

SYS 828

Wir sollten an dieser Stelle vielleicht noch einmal den SYS-Befehl erläutern. Er hat die Aufgabe Maschinenprogramme oder Maschinenroutinen aufzurufen. Diese Programme wurden vorher entweder von Hand aus (so wie in unserem Beispiel) über die Tastatur eingegeben oder aber von Kassette bzw. Diskette geladen. Wenn man diesen Befehl aufruft, teilt man BASIC damit mit, daß die Kontrolle über den Programmablauf dem Maschinenprogramm überlassen wird. Man kann also kein Maschinenprogramm durch ein einfaches RUN starten (es sei denn es existiert eine BASIC-Zeile mit einem SYS) da in diesem Falle der BASIC-Interpreter die Kontrolle über das Maschinenprogramm übernehmen würde. Die Folge wäre irgend ein seltsamer ERROR.

Dieser SYS-Befehl kann aber auch seine Tücken haben. Bei falscher Angabe der Adresse kann es vorkommen, daß der VC-20 abstürzt, das heißt, daß Sie keine Kontrolle mehr über das Gerät haben - da hilft nur noch abschalten. Ein Absturz kann auch erfolgen wenn das Maschinenprogramm fehlerhaft sein sollte. Also aufgepaßt !

Der SYS-Befehl verlangt natürlich auch die Angabe einer Adresse (wie oben schon berichtet). Diese Adresse kann theoretisch Werte zwischen 0 und 65535 anehmen. Diese Adresse gibt die Startadresse des Maschinenprogramms oder die Einsprungadresse der Maschinenroutine an. Diese Maschinenroutine muß mit einem RTS-Befehl (bzw. dem entsprechenden Opcode) abschließen. Ansonsten: ABSTURZ.

übrigens können Sie auch Routinen aus dem ROM aufrufen. Versuchen Sie doch mal den Bildschirm zu löschen ohne die CLR/HOME Taste zu betätigen. Ein Tip: Diese Routine haben wir schon in unserem kleinen Beispielsprogramm verwendet.

DER NÄCHSTE SCHRITT - DIE ASSEMBLERPROGRAMMIERUNG

*** WARUM ASSEMBLER ***

Welche Vorzüge hat denn der Assembler gegenüber der Maschinensprache ? Um diese Frage zu beantworten, muß man sich schon eine gewisse Zeit mit der Maschinensprache und ihren Schwierigkeiten befaßt haben. Man stellt dann nämlich fest, daß die ewige Umrechnung der Opcodes und der Sprünge - vor allen Dingen der indirekten Sprünge - viel zu zeitaufwendig ist. Durch diese Um- und Berechnungen verliert man doch sehr viel Zeit - und der Anfänger vielleicht auch irgendwann einmal die Lust.

Trotzdem möchte man nach den ersten Erfahrungen mit der Maschinenprogrammierung ungern auf ihre Vorzüge, besonders ihre Geschwindigkeit, nicht mehr verzichten.

Es muß doch möglich sein, sich diese ganze Handarbeit zu sparen. Auf diese Idee baut der Assembler auf. Bei ihm ist die Verwendung von symbolischen LABEL möglich. Diese Label sind mit Namensschilder zu vergleichen, die unabhängig von der eigentlichen Adresse sind.

Um diesen Vorgang noch etwas zu verdeutlichen:

In Maschinensprache müßte man sagen: JSR $4711

in Assembler dagegen: JSR EAU_DE_COLOGNE

Wenn man nun irgendwo in dem Maschinenprogramm eine Zeile einfügen möchte, so verändern sich unter Umständen eine ganze Reihe von solchen Adressen. 4711 wird dann vielleicht zu 4714. Und diese neue Adresse muß dann wiederum von Hand überall verändert werden. Eine nette Beschäftigung !

In Assembler dagegen wird diese Arbeit zu einem Kinderspiel. Sie fügen ganz einfach die neue Zeile in das Programm ein, assemblieren erneut und schon haben sich alle Adressen entsprechend verändert. Was für ein Wunder !

Das Assemblerprogramm wird zwar etwas länger, da ja die ganzen Adressen erst in einer Liste mit den entsprechenden Werten eingetragen werden müssen, diese DEFINITIONEN belegen ja einen gewissen Speicherplatz, aber nach dem Assemblieren ist dieses Programm genau so groß wie das eigentliche Maschinenprogramm. Das ist kein Kunststück: Der Assembler mach aus dem Programm ein lauffähiges Maschinenprogramm. Der Zeitaufwand für den Programmier wird daher drastisch verkürzt. Nach der Assemblierung haben Sie dann 2 Programme: Das normale Assemblerprogramm (in dem Sie beliebig viele Änderungen machen können) und das Maschinenprogramm, das Sie nicht mehr zu ändern brauchen.

So wie der Monitor praktisch unabdingbare Voraussetzung für die einfache Maschinenprogrammierung ist, so braucht man einen Assembler, wenn man häufig längere Maschinenprogramme schreiben will. Es sind zur Zeit einige sehr leistungsstarke Monitore und Assembler für den VC-20 erhältlich.

*** TABELLE DER 6502 BEFEHLE ***

In dieser Tabelle zeigen wir Ihnen alle 6502 Befehle. Diese Befehle geben Sie entweder als tatsächlichen Befehl (d.h. die Mnemonics) mit dem unter Umständen dazugehörigen Operanden mit Hilfe eines Assemblers ein, oder aber Sie verwenden die binäre Form für den Monitor (es muß dann natürlich noch eine Umrechnung in Hex-Werte erfolgen).

In unserer Liste sehen Sie viele Binärzahlen die eines oder mehrere kleine 'b's enthalten. Diese 'b' stehen für ein noch zu berechnendes Bit. Da einige Befehle mehrere unterschiedliche Operanden haben können, können auch verschiedene Binärzahlen entstehen. Die Stelle der Binärzahl, die sich unter Umständen ändert, ist aus diesem Grund mit einem 'b' versehen.

Befehl	Funktion	Binär
ADC	Addieren mit übertrag	011bbb01
AND	Logisches UND	001bbb01
ASL	Arithmetisches Linksschieben	000bbb01
BCC	Verzweigen, wenn übertrag gelöscht	10010000
BCS	Verzweigen, wenn übertrag gesetzt	10110000
BEQ	Verzweigen, wenn Ergebnis gleich	11110000
BIT	Teste Bit	0010b100
BMI	Verzweigen, wenn negativ	00110000
BNE	Verzweigen, wenn Ergebnis ungleich	11010000
BPL	Verzweigen, wenn positiv	00010000
BRK	Abbruch	00000000
BVC	Verzweigen, wenn überlauf gelöscht	01010000
BVS	Verzweigen, wenn überlauf gesetzt	01110000
CLC	übertrag löschen	00011000
CLD	Dezimal-Flagge löschen	11011000
CLV	überlauf löschen	10111000
CMP	Vergleich mit Akku	110bbb01
CPX	Vergleich mit X-Register	1110bb00
CPY	Vergleich mit Y-Register	1100bb00
DEC	Dekrementiere Speicher	110bb110
DEX	Dekrementiere X-Register	10101010
DEY	Dekrementiere Y-Register	10001000
EOR	Exklusives ODER	010bbb01
INC	Inkrementiere Speicher	111bb110
INX	Inkrementiere X-Register	11101000
INY	Inkrementiere Y-Register	11001000
JMP	Verzweigen	01b01100
JSR	Verzweigen in Unterprogramm	00100000
LDA	Lade Akku	101bbb01
LDX	Lade X-Register	101bbb10
INY	Lade Y-Register	101bbb00
LSR	Logisches Rechtsschieben	010bbb10
NOP	Leerbefehl (keine Operation)	11101010
ORA	Logisches ODER	000bbb01
PHA	Akku auf Stapel bringen	01001000
PHP	Statusregister auf Stapel bringen	00001000
PLA	Akku vom Stapel holen	01101000
PLP	Statusregister von Stapel holen	00101000
ROL	Linksrotieren	001bbb10

ROR	Rechtsrotieren	011bbb10
RTI	Rückkehr von Unterbrechung	01000000
RTS	Rückkehr von Unterprogramm	01100000
SBC	Subtrahieren mit übertrag	111bbb01
SEC	übertrag setzen	00111000
SED	Dezimal-Flagge setzen	11111000
SEI	Unterbrechung-Flagge setzen	01111000
STA	Akku speichern	100bbb01
STX	X-Register speichern	100bb110
STY	Y-Register speichern	100bb100
TAX	Akku ins X-Register übertragen	10101010
TAY	Akku ins Y-Register übertragen	10101000
TSX	Stapelzeiger ins X-Register übertragen	10111010
TXA	X-Register in Akku übertragen	10001010
TXS	X-Register in Stapelzeiger übertragen	10011010
TYA	Y-Register in Akku übertragen	10011000

Für eine detailiertere Angabe über alle Befehle des 6502 und deren Anwendung und Verwendungszwecke lesen Sie bitte in den entsprechenden 6502-Handbüchern nach.

*** MEIN ERSTES ASSEMBLERPROGRAMM ***

Der Assembler ist ein Programm, das die mnemonische Befehlswiedergabe in ihre binäre Form übersetzt. Jeder symbolische Befehl wird so in einen binären Befehl mit 1, 2 oder 3 Byte Länge übersetzt. So wird aus einem QUELLPROGRAMM (oder Sourcecode) ein OBJEKTPROGRAMM (oder Objectcode). Dieses Programm ist dann genau wie das normale Maschinenprogramm voll lauffähig.

Eine Zeile eines Assemblerprogrammes besteht aus mehreren Teilen:

1.) Die Zeilennummer - Zu vergleichen mit BASIC
2.) Die Adresse - Die Speicherstelle in der sich der Befehl befindet
3.) Der Opcode - Der Befehl hexa-dezimal umgesetzt
4.) Das Label - Der "Titel" eines Programmabschnitts
5.) Die Mnemonics - Der erste Teil des Programms
6.) Der Operand - Der zweite Teil des Programms
7.) Die Bemerkung - Ein beliebiger Text

In dieser Liste sind die Punkte 4 und 7 optional. Das heißt, daß sie nicht in jeder Zeile angegeben werden müssen. Die Adresse braucht nicht mit angegeben zu werden - das ist ja gerade der Vorteil vom Assembler - sie wird während des Assemblierens selber berechnet und eingefügt.

Das fertige Objektprogramm enthält dann nur noch diese Werte. Dadurch ist es möglich, daß die Objektprogramme um einiges kürzer sind als das eigentliche Quellprogramm.

Zusätzlich zu den schon beschriebenen 6502 Befehlen verstehen die meisten Assembler auch noch einige spezielle Befehle. So lassen sich zum Beispiel durch den .TEXT Befehl ganze Texte in den Speicher bringen, ohne daß man ihre Länge berechnen muß. Die selbe Möglichkeit haben wir auch bei Konstanten oder Variablen die mit dem Befehl .BYTE in einen bestimmten Speicherplatz gespeichert werden.

Mit diesen Hilfen werden Sie rasch die Vorzüge der Assemblerprogrammierung erkennen und auch selber sehr viele Routinen, an deren Lösung Sie in BASIC bisher scheiterten, schreiben. In diesem Handbuch werden Sie noch einige Maschinenprogramme finden, die Ihnen auf leicht verständliche Art und Weise zeigen wie man effizient und erfolgreich in Assembler programmiert.

Hier sehen Sie nun ein Ausschnitt aus einem Assemblerprogramm:

absolute Adresse	symbolische Adresse	Befehl	Operanden	Kommentar
0000	PUFFER	=	$033C	Die Variab. PUFFER wird bei $033C gespeichert
0000		* =	0000	Start bei 0000
0000	TEST	LDA	#$00	Die Zahl 0 wird in

				den Akku geladen
0003		STA	PUFFER	Der Akku wird bei $033C gespeichert
0006		JMP	WEITER	Weiter geht es bei WEITER

An dieser Stelle vom Programm können noch weitere Befehle folgen

0123	WEITER	SEC		Die Übertrag-Flagge wird gesetzt
0124		SBC	#$0A	Subtrahiere 10 zum Akku

Ab hier geht das Programm dann weiter.

Sie sehen deutlich den Unterschied zu dem Maschinenprogramm: Sie brauchen nicht mehr die Hex-Codes zu berechnen und einzusetzen. Dies geschieht während des Assemblervorgangs. Gleichzeitig bemerkt er mögliche Fehler und meldet diese dann dem Anwender.

Diese Methode ist gegenüber der Maschinenprogrammierung nicht nur wesentlich schneller sondern auch wesentlich sicherer als die Maschinenprogrammierung.

Nun aber zu unserem ersten selbst erarbeiteten Assemblerprogramm. Als Beispiel soll hier eine mathematische Routine dienen: Eine schnelle SQR (Wurzel) Routine.

Diese Routine ist um einiges schneller und genauer als die im BASIC verwendete SQR-Routine. Der Algorithmus für die Funktion lautet:

```
X(N+1) = X(N) - F(X(N)) / F(X(N))
X = F(A)
X(N+1) = (X(N) + A / X(N)) / 2
```

Die Laufzeit dieser Funktion beträgt nur etwa 14 ms !

Das Programm sieht folgendermaßen aus:

```
100:  033C                    .OPT P1,O2
                  ;
                  ; SCHNELLE SQR-ROUTINE, CA 14 MS
                  ; ALGORITH. X(N+1) = X(N) - F(X(N)) / F(X(N))
                  ; X = F(A) ,  X(N+1) = (X(N) + A / X(N)) / 2
                  ;
                  ; DEKLARATION:
                  ;
110:  033C        SIGN      =   $DC2B
120:  033C        ILLEGAL   =   $D248
130:  033C        EXP       =   $61
140:  033C        AKKU3     =   $57
150:  033C        AKKU4     =   $5C
160:  033C        COUNT     =   $67
170:  033C        A1TOA3    =   $DBCA
180:  033C        A1TOA4    =   $DBC7
190:  033C        MEMDIV    =   $DB0F
200:  033C        MEMPLUS   =   $D867
```

```
210:  033C                     *=   $033C     ;PRG. IST IM BUFFER
220:  033C 20 2B DC            JSR  SIGN      ;VORZEICHEN LESEN
230:  033F F0 34               BEQ  ENDE      ;NULL ? JA ! ENDE
240:  0341 10 03               BPL  OK        ;POSITIVE ? JA ! OK
250:  0343 4C 48 D2            JMP  ILLEGAL   ;ILLEGAL QUANT. ERR.
260:  0346 20 C7 DB   OK       JSR  A1TOA3    ;AKKU1 -> AKKU4
270:  0349 A5 61               LDA  EXP       ;EXPONENT -> AKKU
280:  034B 38                  SEC            ;SETZE ÜBERL. FLAGGE
290:  034C E9 81               SBC  #$81      ;EXP. NORMALISIEREN
300:  034E 08                  PHP            ;STAT.REG. -> STAPEL
310:  034F 4A                  LSR            ;EXPONENT HALBIEREN
320:  0350 18                  CLC            ;Lö. ÜBERLAUF FLAGGE
330:  0351 69 01               ADC  #$01      ;ADDIERE 1 HINZU
340:  0353 28                  PLP            ;STAT.REG. <- STAPEL
350:  0354 90 02               BCC  S1        ;ÜBERL. ? NEIN ! S1
360:  0356 69 7F               ADC  #$7F      ;ADDIERE 127
370:  0358 85 61      S1       STA  EXP       ;SPEICHER ALS EXP.
380:  035A A9 04               LDA  #$04      ;VIER ITERATIONEN
390:  035C 85 67               STA  COUNT     ;SPEICHER IN VAR.
400:  035E 20 CA DB   ITER     JSR  A1TOA3    ;AKKU1 -> AKKU3
410:  0361 A9 5C               LDA  #<AKKU4   ;LSB VON AKKU4
420:  0363 A0 00               LDY  #>AKKU4   ;MSB VON AKKU4
430:  0365 20 0F DB            JSR  MEMDIV    ;DURCH AKKU1 DIV.
440:  0368 A9 57               LDA  #<AKKU3   ;LSB VON AKKU3
450:  036A A0 00               LDA  #>AKKU3   ;MSB VON AKKU3
460:  036C 20 67 D8            JSR  MEMPLUS   ;MIT AKKU1 ADD.
470:  036F C6 61               DEC  EXP       ;AKKU1 / 2 (EXP.-1)
480:  0371 C6 67               DEC  COUNT     ;EINE ITERAT. WENIGER
490:  0373 D0 E9               BNE  ITER      ;WEIT. ITERAT. ? JA !
500:  0375 60         ENDE     RTS            ;ZURÜCK INS PROGRAMM
```

(LSB = Least Significant Byte: b0 - b7)
(MSB = Most Significant Nibble : b8 - b15)

Wenn Sie dieses Programm entweder mit dem Monitor, oder mit einen Assembler eingeben, können Sie es nur auf Diskette abspeichern, da der Kassetten-Puffer durch das Programm selber benutzt wird.

Dieses Programm können Sie nun von BASIC aus verwenden. Sie müssen dazu dem BASIC Programm mitteilen, an welcher Adresse des Speichers sich das Maschinenprogramm befindet. Im BASIC Programm werden dann die zu berechnenden Parameter mit Hilfe der USR-Funktion an das Maschinenprogramm übergeben.

Hier ein Beispiel:

```
10 POKE 1,60: REM LSB - 60 DECIMAL = 3C HEX
20 POKE 2,03: REM MSB - 03 DECIMAL = 03 HEX
30 A=10
40 B=USR(A)
50 PRINT "WURZEL VON ";A;" IST ";B
60 END

RUN (wird über Tastatur aus eingegeben)
```

Die Ausgabe lautet dann:

DIE WURZEL VON 10 IST 3.16227766

Auf diese Art und Weise können Sie sehr leicht eigene mathematische Routinen schreiben, oder schon bestehende (so wie die Funktion SQR) verbessern.

Schreiben Sie doch mal eine Routine, die die Fakultäts-Funktion simuliert, oder ein Maschinenprogramm zur Polynomberechnung.

Grundsätzlich läuft die Erstellung eines Assemblerprogramms nach folgendem Muster ab:

1.) Der Benutzer entwirft mit dem Editor das Quellprogramm.
2.) Der Assembler macht daraus das Zielprogramm.
3.) Dieses Zielprogramm ist dann als Maschinenprogramm lauffähig.

Auf den nächsten Seiten erfahren Sie noch mehr über Assemblerprogrammierung. Wir können Ihnen in diesem Buch nur Denkanstöße geben. Erfahrung sammeln, können Sie nur durch viel, viel Übung. Dazu haben Sie aber ein Gerät das nicht nur ein trockener Computer ist (gibt es sowas überhaupt ?), sondern auch ein hervorragender persöhnlicher Rechner.

*** NOCH MEHR ÜBER DEN ASSEMBLER ***

An dieser Stelle wollen wir Ihnen nun eine kleine Routine vorstellen, die es ermöglicht, den Inhalt des Akkumulators auf dem angeschlossenen Ausgabegerät auszugeben. Wenn Sie also Ihren Drucker als Ausgabegerät geöffnet haben (der BASIC Befehl lautet OPEN 4,4) wird die Ausgabe des Zeichens auch auf dem Drucker erfolgen. Dieser Druck kann natürlich nur dann erfolgen, wenn dieses Zeichen zu Drucken ist. Wie Sie wissen, gibt es ja verschiedene Kontrollzeichen die keine, oder eine fehlerhafte Ausgabe liefern.

Auch dieses Programm steht wieder im Tastaturpuffer. Dadurch können Sie es zwar nicht auf Kassette abspeichern, aber nach Eingabe von Tastatur kann das Programm trotzdem gestartet werden.

```
033C                          .OPT P2,01      ;ASSEMBLERANWEISUNG
033C                          *=   $033C      ;STARTADRESSE (828)
033C               BSOUT      =    $FFD2      ;ZEICHEN-AUSGABE
                              ;ASCII NACH HEX UMWANDLUNG
033C  C9 3A        ASCHEX     CMP  #$3A       ;ZEICHEN = ? !
033E  08                      PHP             ;SICHER ERGEBNIS
033F  29 0F                   AND  #$0F       ;BLENDE b7 BIS b4 AUS
0341  28                      PLP             ;HOLE ERGEBNIS
0342  90 02                   BCC  WROB       ;4 BYTES WEITER
0344  69 08                   ADC  #$08       ;ADD. AKKU+ÜBERTRAG+8
0346  60                      RTS             ;INS HAUPTPROGRAMM
                              ;EIN BYTE AUSGEBEN
0347  48           WROB       PHA             ;RETTE AKKU
0348  4A                      LSR             ;VERSCHIEBE b7 BIS b4
0349  4A                      LSR             ; NACH b3 BIS b0
034A  4A                      LSR             ; UND FÜLLE b7 BIS b4
034B  4A                      LSR             ; MIT 0
034C  20 52 03                JSR  HEXASC     ;SPRINGE ZUR UNTERROUTINE
034F  68                      PLA             ;HOLE AKKU ZURÜCK
0350  29 0F                   AND  #$0F       ;BLENDE b7 BIS b4 AUS
0352  18           HEXASC     CLC             ;LÖSCHE DEN ÜBERTRAG
0353  69 F6                   ADC  #$F6       ;ADD. AKKU+ÜBERLAUF+246
0355  90 02                   BCC  * + 4      ;4 BYTES WEITER
0357  69 06                   ADC  #$06       ;ADD. AKKU+ÜBERLAUF+6
0359  69 3A                   ADC  #$3A       ;ADD. AKKU+ÜBERLAUF+58
035B  4C D2 FF                JMP  BSOUT      ;DRUCKE AKKU
```

Wir wollen nun dieses kleine Programm genau analysieren. Die erste Routine (ASCHEX) wandelt ein ASCII-Zeichen in einen Hex-Wert um und kehrt dann an die Stelle zurück, von der sie aufgerufen wurde.

Die zweite Routine (WROB) ist eigentlich der Hauptteil dieses kleinen Programmes. Wenn Sie eine Hex-Zahl ausgeben wollen, können Sie natürlich nicht beide Zeichen auf einmal ausgeben (z.B. die Zahl FE = 254). Jede Ausgabeeinheit, sei es ein Drucker oder der Bildschirm, kann nur Zeichen für Zeichen ausgeben. Das bedeutet aber, daß diese Zahl in zwei Zahlen aufgeteilt und aufbereitet werden muß. In unserem Beispiel würden wir diese Zahl also in die Zeichen F und E aufspalten.

Dazu teilen wir diese 8-Bit Zahl (8 Bit = 1 Byte kann bekanntlich Zahlen zwischen 0 und 255 darstellen) in jeweils zwei Hälften. Dies geschieht, indem wir die linken 4 Bits der Zahl FE, also die Zahl F, nach rechts verschieben und den freigewordenen Raum mit 0 füllen. Unabhängig von unserem Beispiel haben Sie dann <u>auf jeden Fall</u> eine Zahl zwischen 0 und F.

Der Befehl zur Verschiebung um ein Bit lautet: LSR. Da wir aber 4 Bits verschieben müssen, wenden wir den Befehl auch 4 mal an.

Durch dieses Verschieben würden Sie aber die rechte Seite der Zahl (also das E), einfach überschreiben und damit verlieren. Um das zu umgehen, müssen wir die Zahl irgendwo zwischenspeichern. Die beste Möglichkeit bietet dafür der sogenannte Stapel. Die Zahl E kann dann überschrieben werden, ohne aber verloren zu gehen. Der Befehl für diese Operation lautet: PHA.

Bevor wir also die 4 Bits verschieben, retten wir erst die ganze Zahl (FE). Sehen Sie sich dazu die ersten Zeilen der WROB-Routine einmal an.

Nun haben Sie nur noch ein Zeichen (das F). Dieses Zeichen könnte nun direkt ausgegeben werden. Dazu muß es aber erst noch aufbereitet werden, da das Zeichen mit dem Code 15 (=F) nicht gleich dem Zeichen F ist. Sie können das ja ganz einfach nachprüfen, indem Sie sich von BASIC aus den CHR$ von 15 ausgeben lassen: PRINT CHR$(15). Wie Sie sehen, sehen Sie außer der Meldung READY gar nichts. So einfach ist das mit der Ausgabe also nicht. Wir müssen das Zeichen erst aufbereiten.

Dazu rufen wir das Unterprogramm HEXASC auf, das aus einer Hex-Zahl ein ASCII-Zeichen macht. Zunächst wird das Überlauf-Flag gelöscht, damit die nächsten Additionen korrekt ausgeführt werden. In diesem Zusammenhang wollen wir noch erklären, was eigentlich ÜBERLAUF bedeutet. Ganz einfach gesagt: Eine Zahl ist größer als 255. Sie übersteigt also das Fassungsvermögen eines Bytes. Diese Flagge wird mit dem Befehl CLC also gelöscht. Das hat einen ganz bestimmten Grund. Sie verarbeiten ja bekanntlich Zahlen zwischen 0 und 9 und Buchstaben von A bis B. Wenn Sie nun in der Reihenfolge der Hex-Werte (0-F), auch die entsprechenden Zeichen bilden würden, so wären die Resultate ab dem Hex-Wert A falsch. Denn in der Zeichentabelle, in der die 'Übersetzung' von Hex nach ASCII durchgeführt wird, befinden sich zwischen '9' und 'A' noch einige andere Sonderzeichen.

Wir müssen also in der Übersetzung einen 'Sprung' machen. Aus diesem Grund addieren wir die Zahl 246.

Haben wir eine Zahl <= 9 so ist das Resultat dieser Addition kleiner oder gleich 255. Die Überlauf Flagge wird also nicht gesetzt. Den ersten Teil, von 0 bis 9, können wir nun leicht am nicht gesetzten Überlauf erkennen. Nun können wir auch entscheiden, wo in der Zeichentabelle unser Zeichen liegt. Durch die Abfrage ob die Überlauf Flagge gestzt ist oder nicht, können wir an zwei verschiedene Stellen verzweigen.

Ist das Bit nicht gesetzt, addieren wir noch einmal die Zahl 58. Der kleinste Wert wäre dabei die Zahl 48 (Überlauf-Flagge ist nun natürlich gesetzt, aber das spielt keine Rolle mehr) oder aus der Zeichentabelle das Zeichen 0. Der größte Wert ist die Zahl 57 oder das Zeichen 9.

War die Überlauf-Flagge aber im ersten Fall gesetzt, werden zu den 58 noch einmal 6 dazu addiert. Diese 6 machen genau den Sprung aus, den wir in der Zeichentabelle überbrücken müssen. Das Ergebnis dieser Addition liefert uns dann einen Wert zwischen 65 (das Zeichen dafür ist A) und 70 (dem Zeichen F). Mit diesem Ergebnis können wir nun zu der Ausgaberoutine gehen und das Zeichen ausgeben.

Diese Ausgaberoutine brauchen wir nicht selber zu schreiben. Es existiert bereits im ROM des VC-20 eine Routine die uns diese Arbeit abnimmt. Diese Routine finden wir ab der Adresse $FFD2 oder aber unter dem Namen BSOUT in dem ROM-LISTING.

Nach der Ausgabe dieses ersten Zeichen (in unserem Beispiel das Zeichen F) können wir nun auch die zweite Zahl ausgeben. Hierzu müssen wir nun zunächst unseren ursprünglichen Wert (FE) wieder vom Stapel zurückholen. Denn in der Zwischenzeit haben wir ja das Zeichen E nicht mehr zur Verfügung. Mit dem Befehl PLP können wir die Zahl vom Stapel wieder in den Akku bringen.

Nun gehen wir ähnlich der Ausgabe bei dem ersten Zeichen vor. Zunächst muß die linke Hälfte ausgeblendet werden (eben haben wir durch Verschieben die rechte Hälfte quasi ausgeblendet). Dieses Ausblenden geschieht durch eine einfache UND-Verknüpfung mit der Zahl F. Die Zahl F sieht binär so aus:

0 0 0 0 1 1 1 1

Bei einer UND-Verknüpfung wird jedes Bit nur dann auf 1 gesetzt, wenn beide Bits auf 1 gesetzt sind. Diesen Vorgang kann man sich graphisch am besten veranschaulichen:

```
                   FE : 1 1 1 1 1 1 1 0
verknüpft mit      0F : 0 0 0 0 1 1 1 1

ergibt             0E : 0 0 0 0 1 1 1 0
```

So läßt sich also sehr leicht diese Zahl in den rechten Teil verwandeln. Der Befehl für diese UND-Verknüpfung mit F lautet: AND #$0F.

Die Zahl steht nun wieder im richtigen Format und wir können Sie ausgeben. Dazu verwenden wir natürlich wieder unsere eben schon besprochene Routine.

Dieses vollständige Programm soll Ihnen einen Eindruck über die Wirkungsweise des Assemblers geben. Auch hier können wir nur den Tip geben: Lassen Sie alles Gelernte noch einmal an sich vobeilaufen - wenn Sie etwas nicht verstanden haben, lesen Sie es ruhig noch ein zweites, drittes mal. Dann klappt es bestimmt.

Als Abschluß dieses Kapitels wollen wir Ihnen noch ein Programm vorstellen, mit dem Sie die vorgestellten Maschinenprogramm wenigstes eingeben können. Es ersetzt zwar keinen Monitor, Sie können aber die Opcodes direkt in Hex-Form eingeben, ohne diese erst in Dezimal umrechnen zu müssen.

```
100 AD=828: STARTADRESSE DES MASCHINENPROGRAMMES (DEZIMAL)
105 AN=34: REM AN=ANZAHL DER OPCODES
110 FOR I=0 TO AN-1
120 READ OP$
130 GOSUB 60000
140 POKE AD+I,OP
150 NEXT I
160 END
170 REM DIE OPCODES KöNNEN NUN IN FORM VON DATA ZEILEN EINGEGEBEN
180 REM WERDEN. ALS BEISP. HABEN WIR DAS AUSGABEPROGR. GENOMMEN,
190 REM WAS WIR OBEN AUSFüHRLICH BESPROCHEN HABEN.
200 DATA C9,3A,08,29,0F,28,90,02
210 DATA 69,08,60,48,4A,4A,4A,4A
220 DATA 20,52,03,68,29,0F,18,69
230 DATA F6,90,02,69,06,69,3A,4C
240 DATA D2,FF
60000 REM ROUTINE ZUR UMWANDLUNG VON HEX NACH DEZIMAL
60010 QL$=LEFT$(OP$,1): QR$=RIGHT$(OP$,1)
60020 QL=VAL(QL$): IF QR=0 AND QR$<>"0" THEN QL=ASC(QR$)-55
60030 QR=VAL(QR$): IF QR=0 AND QR$<>"0" THEN QR=ASC(QR$)-55
60040 OP=QL*16+QR
60050 RETURN
```

Mit diesem Programm können Sie nun Byte für Byte eingeben. Allerdings werden Sie recht schnell feststellen, daß Ihnen diese Art der Maschinenprogrammierung nicht mehr ausreicht. Dann kommen Sie automatisch auf einen Monitor zurück.

*** BASIC - MASCHINENSPRACHE - ASSEMBLER ***

Wir haben in diesem Buch schon oft Verbindungen zwischen BASIC und Maschinensprache / Assembler angesprochen. Was passiert dabei eigentlich und wie funktionieren die verschiedenen Befehle ?

1.) NEW

Ein Befehl der unter Umständen fatale Folgen haben kann - aber was macht er denn nun tatsächlich ? Wir haben oben schon seine Wirkung nach Laden von Maschinenprogrammen angesprochen. Alle Register werden wieder auf den Urzustand gesetzt - ein eventuell im Speicher befindliches BASIC-Programm wird natürlich gelöscht.

2.) PEEK

Mit dieser Funktion können Sie alle Speicherstellen auslesen. Die erhaltenen Werte sind in dezimaler Form. Zur Maschinenprogrammierung empfiehlt sich daher eine Konvertierung dieser Werte in Hex-Zahlen.

3.) POKE

Durch den POKE-Befehl werden einzelne Speicherstellen mit dem angegebenen Wert gefüllt. Auch hier erfolgt die Angabe wieder in dezimaler Form. Mit unserem Beispielsprogramm haben wir aber eine Möglichkeit geschaffen, auch Hex-Werte zu verarbeiten.

4.) SYS

Dieser Befehl startet das, an der angegebenen Speicherstelle stehende, Maschinenprogramm. Es können allerdings keine PARAMETER (Werte) übergeben werden.

5.) USR

Mit dieser Funktion rufen Sie ein Maschinenprogramm auf, dessen Startadresse Sie vorher dem Computer mitgeteilt haben. Beim VC-20 sind dies die Adressen 1 und 2. (s. SQR-Routine) Der Vorteil gegenüber dem SYS-Befehl liegt in der Möglichkeit Parameter zu übergeben.

Die Belegung der Zeropage und weiterer wichtiger Bereiche

Hexadresse	Dezimal	Belegung
00	0	JMP-Befehl $4C für USR-Funktion
01 - 02	1 - 2	$D248 USR-Vektor
03 - 04	3 - 4	Vektor für Umwandlung Fließkomma nach Fest
05 - 06	5 - 6	Vektor Umwandlung Fest nach Fließkomma
07	7	Suchzeichen
08	8	Hochkomma-Flag
09	9	Speicher für Spalte beim TAB-Befehl
0A	10	Load = 0, Verify = 1, Flag des Interpreters
0B	11	Zeiger in Eingabepuffer, Anzahl der Dimensionen
0C	12	Flag für DIM
0D	13	Typflag $00 = numerisch, $FF = String
0E	14	Flag für Integer = $80, Real = $00
0F	15	Hochkomma-Flag bei LIST
10	16	Flag für FN
11	17	Flag für INPUT $00, GET $40, READ $98
12	18	Vorzeichen bei ATN
13	19	aktives I/O-Gerät
14 - 15	20 - 21	Integer-Adresse, z.B. Zeilennummer
16	22	Zeiger auf Stringstack
17 - 18	23 - 24	Zeiger auf zuletzt verwendeten String
19 - 21	25 - 33	Stringstack
22 - 25	34 - 37	Zeiger für diverse Zwecke
26 - 2A	38 - 42	Register für Funktionsauswertung und Arithmetik
2B - 2C	43 - 44	Zeiger auf BASIC-Programmstart
2D - 2E	45 - 46	Zeiger auf Start der Variablen
2F - 30	47 - 48	Zeiger auf Start der Arrays
31 - 32	49 - 50	Zeiger auf Ende der Arrays
33 - 34	51 - 52	Zeiger auf Beginn der Strings
35 - 36	53 - 54	Hilfszeiger für Strings
37 - 38	55 - 56	Zeiger auf BASIC-RAM Ende
39 - 3A	57 - 58	augenblickliche BASIC-Zeilennummer
3B - 3C	59 - 60	vorherige BASIC-Zeilennummer
3D - 3E	61 - 62	Zeiger auf nächstes BASIC-Statement für CONT
3F - 40	63 - 64	augenblickliche Zeilennummer für DATA
41 - 42	65 - 66	Zeiger auf nächstes DATA-Element
43 - 44	67 - 68	Zeiger auf Herkunft der Eingabe
45 - 46	69 - 70	Variablenname
47 - 48	71 - 72	Variablenadresse
49 - 4A	73 - 74	Variablenzeiger für FOR/NEXT
4B - 4C	75 - 76	Zwischenspeicher für Programmzeiger
4D	77	Maske für Vergleichoperationen
4E - 4F	78 - 79	Zeiger für FN
50 - 53	80 - 83	Stringdescriptor
54	84	Konstante $4C JMP für Funktionen
55 - 56	85 - 86	Sprungvektor für Funktionen
57 - 5B	87 - 91	Register für Arithmetik, Akku#3
5C - 60	92 - 96	Register für Arithmetik, Akku#4
61 - 65	97 - 101	Fließkommaakku#1, FAC
66	102	Vorzeichen von FAC
67	103	Zähler für Polynomauswertung
68	104	Rundungsbyte für FAC
69 - 6D	105 - 109	Fließkommaakku#2, ARG
6E	110	Vorzeichen von ARG
6F	111	Vergleichsbyte der Vorzeichen von FAC und ARG
70	112	Rundungsbyte für FAC
71 - 72	113 - 114	Zeiger für Polynomauswertung

Hexadresse	Dezimal	Belegung
73 - 8A	115 - 138	CHRGET - Routine, holt Zeichen aus BASIC-Text
7A - 7B	122 - 123	Programmzeiger
8B - 8F	139 - 143	letzter RND-Wert
90	144	Statuswort ST
91	145	Flag für Stop-Taste
92	146	Zeitkonstante für Band
93	147	Flag für LOAD $00 oder VERIFY $01
94	148	Flag bei IEC-Ausgabe
95	149	Ausgabepuffer für IEC-Bus
96	150	Flag für EOT vom Band empfangen
97	151	Zwischenspeicher für Register
98	152	Anzahl der offenen Files
99	153	aktives Eingabegerät
9A	154	aktives Ausgabegerät
9B	155	Parität für Band
9C	156	Flag für Byte empfangen
9D	157	Flag für Direkt-Modus $80, Programm $00
9E	158	Band Pass 1 Checksumme
9F	159	Band Pass 2 Fehlerkorrektur
A0 - A2	160 - 162	Time
A3	163	Bitzähler für serielle Ausgabe
A4	164	Zähler für Band
A5	165	Zähler für Band schreiben
A6	166	Zeiger in Bandpuffer
A7 - AB	167 - 171	Arbeitspeicher für Bandein/ausgabe
AC - AD	172 - 173	Zeiger für Bandpuffer und Scrolling
AE - AF	174 - 175	Zeiger auf Programmende bei LOAD/SAVE
B0 - B1	176 - 177	Zeitkonstanten für Band-Timing
B2 - B3	178 - 179	Zeiger auf Bandpuffer
B4	180	Bitzähler für Band
B5	181	nächstes Bit für RS 232
B6	182	Puffer für auszugebendes Byte
B7	183	Länge des Filenamens
B8	184	logische Filenummer
B9	185	Sekundäradresse
BA	186	Gerätenummer
BB - BC	187 - 188	Zeiger auf Filenamen
BD	189	Arbeitsspeicher serielle Ein/Ausgabe
BE	190	Passzähler für Band
BF	191	Puffer für serielle Ausgabe
C0	192	Flag für Bandmotor
C1 - C2	193 - 194	Startadresse für Ein/Ausgabe
C3 - C4	195 - 196	Endadresse für Ein/Ausgabe
C5	197	letzte gedrückte Taste
C6	198	Anzahl der gedrückten Tasten
C7	199	Flag für RVS-Modus
C8	200	Zeilenende für Eingabe
C9	201	Cursorzeile für Eingabe
CA	202	Cursorspalte für Eingabe
CB	203	gedrückte Taste, keine Taste = 64
CC	204	Flag für blinkenden Cursor
CD	205	Zähler für Cursor blinken
CE	206	Zeichen unter dem Cursor
CF	207	Flag für Cursor blinken
D0	208	Flag für Eingabe von Tastatur oder Bildschirm
D1 - D2	209 - 210	Zeiger auf Start der aktuellen Bildschirmzeile
D3	211	Cursorspalte
D4	212	Flag für Cursor
D5	213	Länge der Bildschirmzeile

Hexadresse	Dezimal	Belegung
D6	214	Cursorzeile
D7	215	diverse Zwecke
D8	216	Anzahl der Inserts
D9 - F2	217 - 242	MSB der Bildschirmzeilenanfänge
F3 - F4	243 - 244	Zeiger in Farb-RAM
F5 - F6	245 - 246	Zeiger auf Tastatur-Dekodiertabelle
F7 - F8	247 - 248	Zeiger auf RS 232 Eingabepuffer
F9 - FA	249 - 250	Zeiger auf RS 232 Ausgabepuffer

****

00FF - 010A	255 - 266	Puffer für Umwandlung Fließkomma nach ASCII
0100 - 013E	256 - 318	Speicher für Korrektur bei Bandeingabe
0100 - 01FF	256 - 511	Prozessor Stack
0200 - 0258	512 - 600	BASIC Eingabepuffer
0259 - 0262	601 - 610	Tabelle der logischen Filenummern
0263 - 026C	611 - 620	Tabelle der Gerätenummern
026D - 0276	621 - 630	Tabelle der Sekundäradresse
0277 - 0280	631 - 640	Tastaturpuffer
0281 - 0282	641 - 642	Start des BASIC-RAM
0283 - 0284	643 - 644	Ende des BASIC-RAM
0285	645	Timeout-Flag für seriellen IEC-Bus
0286	646	augenblickliche Farbe
0287	647	Farbe unter dem Cursor
0288	648	High-Byte Video-RAM
0289	649	Länge des Tastaturpuffers
028A	650	Flag für Repeatfunktion für alle Tasten
028B	651	Zähler für Repeat-Geschwindigkeit
028C	652	Zähler für Repeat-Verzögerung
028D	653	Flag für Shift und CTRL
028E	654	Shift-Flag
028F - 0290	655 - 656	Zeiger für Tastatur-Dekodierung
0291	657	Flag für Shift/Commodore gesperrt
0292	658	Flag für Scrollen
0293	659	RS 232 Kontrollwort
0294	660	RS 232 Befehlswort
0295 - 0296	661 - 662	Bit-Timing
0297	663	RS 232 Status
0298	664	Anzahl der Datenbits für RS 232
0299 - 029A	665 - 666	RS 232 Baud-Rate
029B	667	Zeiger auf empfangenes Byte RS 232
029C	668	Zeiger auf Input von RS 232
029D	669	Zeiger auf zu übertragendes Byte RS 232
029E	670	Zeiger auf Ausgabe auf RS 232
029F - 02A0	671 - 672	Speicher für IRQ während Bandbetrieb
02A1	673	VIA 2 NMI-Flag
02A2	674	VIA 1 Timer A
02A3	675	VIA 1 Interruptflag
02A4	676	VIA 1 Flag für Timer A
02A5	677	Bildschirmzeile
0300 - 0301	768 - 769	Vektor für Fehlermeldung
0302 - 0303	770 - 771	Vektor für BASIC-Warmstart
0304 - 0305	772 - 773	Vektor für Umwandlung in Interpreterkode
0306 - 0307	774 - 775	Vektor für Umwandlung in Klartext (LIST)
0308 - 0309	776 - 777	Vektor für BASIC-Befehlsadresse holen
030A - 030B	778 - 779	Vektor für Ausdruck auswerten
030C	780	Akku für SYS-Befehl
030D	781	X-Reg für SYS-Befehl
030E	782	Y-Reg für SYS-Befehl

Hexadresse	Dezimal	Belegung
030F	783	Status für SYS-Befehl
0314 - 0315	788 - 789	$EABF IRQ-Vektor
0316 - 0317	790 - 791	$FED2 BRK-Vektor
0318 - 0319	792 - 793	$FEAD NMI-Vektor
031A - 031B	794 - 795	$F40A OPEN-Vektor
031C - 031D	796 - 797	$F43A CLOSE-Vektor
031E - 031F	798 - 799	$F2C7 CHKIN-Vektor
0320 - 0321	800 - 801	$F309 CKOUT-Vektor
0322 - 0323	802 - 803	$F3F3 CLRCH-Vektor
0324 - 0325	804 - 805	$F20E INPUT-Vektor
0326 - 0327	806 - 807	$F27A OUTPUT-Vektor
0328 - 0329	808 - 809	$F770 STOP-Vektor
032A - 032B	810 - 811	$F1F5 GET-Vektor
032C - 032D	812 - 813	$F3EF CLALL-Vektor
0330 - 0331	816 - 817	$F549 LOAD-Vektor
0332 - 0333	818 - 819	$F685 SAVE-Vektor
033C - 03FB	828 -1019	Bandpuffer
0400 - 0FFF	1024 -4095	Raum für 3K-RAM-Erweiterung
1000 - 11FF	4096 -4607	Video-RAM bei mehr als 3K RAM-Erweiterung
1E00 - 1FFF	7680 -8191	Video-RAM bei Grundversion und 3K-Erweiterung
2000 - 7FFF	8192 -32767	Raum für 8 - 24K-RAM/ROM-Erweiterung
8000 - 8FFF	32768-36863	Character-Generator
9000 - 900F	36864-36879	Video-Controller
9110 - 911F	37136-37151	VIA 6522 1
9120 - 912F	37152-37167	VIA 6522 2
A000 - BFFF	40960-49051	Raum für Auto-Start-ROM oder RAM
C000 - FFFF	49052-65535	BASIC-Interpreter und Betriebssystem

Die Routinen des BASIC-Interpreters

In der folgenden übersicht sind die Adressen der Routinen des BASIC-Inte
des VC 20 zusammengestellt. Bei der übertragung von Maschinenprogrammen
20 auf den Commodore 64 ist eine einfache Umrechnung der Adressen mögli
VC 20 Adressen zwischen $C000 bis $DFFF wird einfach $2000 abgezogen,
64er Adresse zu erhalten, bei Adressen ab $E000 wird 3 dazuaddiert. Au
beim VC 20 wird $A3B8 auf dem 64er und aus $E094 wird $E097.

Adresse	Beschreibung
C000	Startvektor
C002	NMI-Vektor
C004	'cbmbasic'
C00C	Adressen der BASIC-Befehle minus 1
C052	Adressen der BASIC-Funktionen
C080	Hierarchiekodes und Adressen der BASIC-Operatoren
C09E	Liste der BASIC-Befehlsworte
C19E	BASIC-Fehlermeldungen
C328	Adressen der Fehlermeldungen
C364	Meldungen des BASIC-Interpreters
C38A	Stapelsuchroutine für FOR-NEXT und GOSUB
C3B8	Blockverschieberoutine
C3FB	prüft auf Platz im Stapel
C408	schafft Platz im Speicher
C435	Ausgabe von 'out of memmory'
C437	Fehlermeldung ausgeben
C469	Break-Einsprung
C474	Ready-Einsprung
C480	Eingabe-Warteschleife
C49C	Löschen und Einfügen von Programmzeilen
C533	BASIC-Programmzeilen neu binden
C560	holt eine Zeile in den Eingabepuffer
C571	Ausgabe von 'string too long'
C579	Umwandlung einer Zeile in Interpreterkode
C613	Startadresse einer BASIC-Zeile suchen
C642	BASIC-Befehl NEW
C65E	BASIC-Befehl CLR
C68E	Programmzeiger auf BASIC-Start setzen
C69C	BASIC-Befehl LIST
C717	Interpreterkode in Befehlswort umwandeln
C742	BASIC-Befehl FOR
C7AE	Interpreterschleife, führt BASIC-Befehle aus
C7ED	führt einen BASIC-Befehl aus
C81D	BASIC-Befehl RESTORE
C82C	bricht Programm bei gedrückter Stop-Taste ab
C82F	BASIC-Befehl STOP
C831	BASIC-Befehl END
C857	BASIC-Befehl CONT
C871	BASIC-Befehl RUN
C883	BASIC-Befehl GOSUB
C8A0	BASIC-Befehl GOTO
C8F8	Basic-Befehl RETURN
C8F8	BASIC-Befehl DATA
C906	sucht nächstes Statement
C909	sucht nächste Zeile
C928	BASIC-Befehl IF
C93B	BASIC-Befehl REM
C94B	BASIC-Befehl ON
C96B	sucht Adresse einer BASIC-Zeile

Adresse	Beschreibung
C9A5	BASIC-Befehl LET
CA80	BASIC-Befehl PRINT#
CA86	BASIC-Befehl CMD
CAA0	BASIC-Befehl PRINT
CB1E	String ausgeben
CB3E	Leerzeichen bzw. Cursor right ausgeben
CB4D	Fehlerbehandlung bei Eingabe
CB7B	BASIC-Befehl GET
CBA5	BASIC-Befehl INPUT#
CBBF	BASIC-Befehl INPUT
CC06	BASIC-Befehl READ
CCFC	'?extra ignored' und '?redo from start'
CD1D	BASIC-Befehl NEXT
CD8A	FRMNUM holt Ausdruck und prüft auf numerisch
CD8D	prüft auf numerisch
CD8F	prüft auf String
CD99	Ausgabe von 'typ mismatch'
CD9E	FRMEVL holt und wertet beliebigen Ausdruck aus
CE83	arithmetischen Ausdruck holen
CEA8	Fließkommakonstante Pi
CED4	BASIC-Befehl NOT
CEF1	holt Ausdruck in Klammern
CEF7	prüft auf 'Klammer zu'
CEFA	prüft auf 'Klammer auf'
CEFD	prüft auf 'Komma'
CEFF	prüft auf Zeichen im Akku
CF08	Ausgabe von 'syntax error'
CF28	holt Variable
CFE6	BASIC-Befehl OR
CFE9	BASIC-Befehl AND
D016	Vergleichsoperationen
D081	BASIC-Befehl DIM
D113	prüft auf Buchstabe
D194	berechnet Zeiger auf erstes Arrayelement
D1A5	Fließkommakonstante -32768
D1AA	FAC nach Integer wandlen
D245	Ausgabe von 'bad subscript'
D248	Ausgabe von 'illegal quantity'
D34C	berechnet Arraygröße
D37D	BASIC-Funktion FRE
D39E	BASIC-Funktion POS
D3A6	Test auf Direkt-Modus
D3AB	Ausgabe von 'illegal direct'
D3AE	Ausgabe von 'undef'd function'
D3B3	BASIC-Befehl DEF
D3E1	FN-Syntax prüfen
D3F4	BASIC-Funktion FN
D465	BASIC-Funktion STR$
D475	Stringverwaltung, Zeiger auf String berechnen
D487	String einrichten
D526	Garbage Collection, nichtgebrauchte Strings entfernen
D63D	Stringverknüpfung '+'
D6A3	Stringverwaltung FRESTR
D6EC	BASIC-Funktion CHR$
D700	BASIC-Funktion LEFT$
D72C	BASIC-Funktion RIGHT$
D737	BASIC-Funktion MID$
D77C	BASIC-Funktion LEN
D782	Stringparameter holen

Adresse	Beschreibung
D7BB	BASIC-Funktion ASC
D79B	Holt Byte-Ausdruck (0 bis 255)
D7AD	BASIC-Funktion VAL
D7EB	Holt Adresse (0 bis 65535) und Byte-Wert (0 bis 255)
D7F7	FAC nach Adressformat wandlen (Bereich 0 bis 65535)
D80D	BASIC-Funktion PEEK
D824	BASIC-Befehl POKE
D82D	BASIC-Befehl WAIT
D849	FAC = FAC + 0.5
D850	Minus FAC = Konstante (A/Y) - FAC
D853	Minus FAC = ARG - FAC
D867	Plus FAC = Konstante (A/Y) - FAC
D86A	Plus FAC = ARG + FAC
D97E	Ausgabe von 'overflow'
D9BC	Fließkommakonstanten für LOG
D9EA	BASIC-Funktion LOG
DA28	Multiplikation FAC = Konstante (A/Y) * FAC
DA2B	Multiplikation FAC = ARG * FAC
DA8C	ARG = Konstante (A/Y)
DAE2	FAC = FAC * 10
DAF9	Fließkommakonstante 10
DAFE	FAC = FAC / 10
DB0F	FAC = Konstante (A/Y) / FAC
DB12	FAC = ARG / FAC
DB8A	Ausgabe von 'division by zero'
DBA2	FAC = Konstante (A/Y)
DBC4	Akku#4 = FAC
DBCA	Akku#3 = FAC
DBD0	Variable = FAC
DBFC	FAC = ARG
DC0C	ARG = FAC
DC1B	FAC runden
DC2B	Vorzeichen von FAC holen
DC39	BASIC-Funktion SGN
DC58	BASIC-Funktion ABS
DC5B	Konstante (A/Y) mit FAC vergleichen
DC9B	Umwandlung FAC nach Integer
DCCC	BASIC-Funktion INT
DCF3	Umwandlung ASCII nach Fließkomma
DDB3	Fließkommakonstanten für Fließkomma nach ASCII
DDC2	Ausgabe der Zeilennumer bei Fehlermeldung
DDCD	Positive Integerzahl (0 bis 65535) ausgeben
DDDD	FAC nach ASCII-Format wandeln
DF11	Fließkommakonstante 0.5
DF16	Binärzahlen für Umwandlung FAC nach ASCII
DF71	BASIC-Funktion SQR
DF78	Potenzierung FAC = Konstante (A/Y) hoch FAC
DF7B	Potenzierung FAC = ARG hoch FAC
DFBF	Fließkommakonstanten für EXP
DFED	BASIC-Funktion EXP
E040	Polynomberechnung
E056	Polynomberechnung
E08A	Fließkommakonstanten für RND
E094	BASIC-Funktion RND
E104	Ausgabe von 'break'
E109	BSOUT ein Zeichen ausgeben
E10F	BASIN ein Zeichen empfangen
E115	CKOUT Ausgabegerät festsetzen
E11B	CHKIN Eingabegerät festsetzen

Adresse	Beschreibung
E121	GETIN ein Zeichen holen
E127	BASIC-Befehl SYS
E153	BASIC-Befehl SAVE
E162	BASIC-Befehl VERIFY
E165	BASIC-Befehl LOAD
E1BB	BASIC-Befehl OPEN
E1C4	BASIC-Befehl CLOSE
E1D1	Parameter für LOAD und SAVE holen
E216	Parameter für OPEN holen
E261	BASIC-Funktion COS
E268	BASIC-Funktion SIN
E2B1	BASIC-Funktion TAN
E2DD	Fließkommakonstanten für SIN und COS
E30B	BASIC-Funktion ATN
E33B	Fließkommakonstanten für ATN
E37B	BASIC-Kaltstart
E387	Kopie der CHRGET-Routine
E39F	Anfangswert für RND-Funktion
E3A4	RAM für BASIC initialisieren
E444	Tabelle der BASIC-Vektoren
E45B	BASIC-Vektoren laden

Die Routinen des VC 20 Betriebssystems

In der folgenden übersicht sind die wichtigsten Adressen des VC 20 Betriebssystems zusammengestellt. Um die übertragung von VC 20-Programmen auf den neuen Commodore 64 zu erleichtern, ist gleichzeitig auch die entsprechende Adresse des Commodore 64 angegeben.

64	VC 20	Beschreibung
E45F	E429	Meldungen des Betriebssystems
E500	E500	holt BASIC-Adresse des CIAs bzw. VIAs
E505	E505	holt Bildschirmformat Zeilen/Spalten
E50A	E50A	Cursor setzen bzw. Cursorposition holen
E518	E518	Bildschirm-Reset
E544	E55F	Bildschirm löschen
E566	E581	Cursor Home
E5A0	E5BB	Videocontroller initialisieren
E5B4	E5CF	Zeichen aus Tastaturpuffer holen
E5CA	E5E5	Warteschleife für Tastatureingabe
E632	E64F	ein Zeichen vom Bildschirm holen
E684	E6B8	testet auf Hochkomma
E6B6	E6EA	MSB für Zeilenanfänge berechnen
E8DA	E921	Tabelle der Farbkodes
E8EA	E975	Bildschirm scrollen
E9C8	EA56	Zeile nach oben schieben
E9FF	EA8D	Bildschirmzeile löschen
EA1C	EAA1	Zeichen und Farbe auf Bildschirm setzen
EA24	EAB2	Zeiger auf Farb-RAM berechnen
EA31	EABF	Interrupt-Routine
EA87	EB1E	Tastaturabfrage
EB48	EBDC	Prüfung auf Shift, CTRL und Commodore-Taste
EB79	EC46	Zeiger auf Tastatur-Dekodiertabellen
EB81	EC5E	Dekodiertabellen
EC44	ED21	Prüfung auf Steuerzeichen
EC78	ED69	Dekodiertabellen
ECB9	EDE4	Konstanten für Videocontroller
ECE7	EDF3	'load (cr) run (cr)'
ECF0	EDFE	Tabelle der LSB der Bildschirmanfänge
ED09	EE14	TALK senden
ED0C	EE17	LISTEN senden
ED40	EEE4	ein Byte auf IEC-Bus ausgeben
EDB9	EEC0	Sekundäradresse für LISTEN senden
EDC7	EECE	Sekundäradresse für TALK senden
EDEF	EEF6	UNTALK senden
EDFE	EF04	UNLISTEN senden
EE13	EF19	ein Byte vom IEC-Bus holen
EEB3	EF96	Verzögerung eine Millisekunde
EEBB	EFA3	RS 232 Ausgabe
EF4A	F027	Anzahl der RS 232 Datenbits berechnen
F014	F0ED	Ausgabe in RS 232 Puffer
F086	F14F	GET von RS 232
F0A4	F160	Timer für IEC-Timeout setzen
F0BD	F174	Fehlermeldungen des Betriebssystems
F12B	F1E0	Meldungen ausgeben
F157	F20E	BASIN ein Zeichen holen
F1CA	F27A	BSOUT ein Zeichen ausgeben
F20E	F2C7	CHKIN festlegen des Eingabegeräts
F250	F309	CKOUT festlegen des Ausgabegeräts
F291	F34A	CLOSE

64	VC 20	Beschreibung
F30F	F3CF	logische Filenummer suchen
F31F	F3DF	Fileparameter setzen
F32F	F3EF	CLALL schließt alle I/O-Kanäle
F34A	F40A	OPEN
F49E	F542	LOAD
F5AF	F647	'searching for filename' ausgeben
F5D2	F66A	'loading/verifying' ausgeben
F5DD	F675	SAVE
F68F	F728	'saving filename' ausgeben
F69B	F734	UDTIM laufende Zeit erhöhen
F6DD	F760	Time holen
F6E4	F767	Time setzen
F6ED	F770	Stop-Taste abfragen
F6FB	F77E	Fehlermeldungen des Betriebssystems ausgeben
F72C	F7AF	Programmheader vom Band lesen
F76A	F7E7	Header auf Band schreiben
F7D0	F84D	Startadresse des Bandpuffers holen
F7D7	F854	Start und Endadresse des Bandpuffers setzen
F7EA	F867	Bandheader nach Namen suchen
F80D	F88A	Bandpufferzeiger erhöhen
F817	F894	wartet auf Bandtaste für lesen
F82E	F8AB	frägt Bandtaste ab
F838	F8B7	wartet auf Bandtaste für schreiben
F841	F8C0	Block vom Band lesen
F84A	F8C9	Programm vom Band laden
F864	F8EA	Bandpuffer auf Band schreiben
F86B	F8EA	Block bzw. Programm auf Band schreiben
F8BE	F92F	I/O-Abschluß abwarten
F8E1	F94B	testet auf Stop-Taste
F92C	F98E	Interrupt-Routine für Band lesen
FB97	FBDB	Bitzähler für serielle Ausgabe setzen
FBA6	FBEA	ein Bit auf Band schreiben
FBCD	FC0B	Interrupt-Routine für Band schreiben
FCB8	FCF6	IRQ-Vektor setzen
FCCA	FD08	Bandmotor ausschalten
FCD1	FD11	prüft auf Erreichen der Endadresse
FCDB	FD1B	Adresszeiger erhöhen
FCE2	FD22	RESET
FD02	FD3F	prüft auf ROM in $8000 bzw. $A000
FD10	FD4D	ROM-Modul Identifizierung
FD15	FD52	Hardware und I/O Vektoren setzen bzw. holen
FD30	FD6D	Tabelle der Hardware und I/O-Vektoren
FD50	FD8D	Arbeitsspeicher initialisieren
FD9B	FD6D	Tabelle der IRQ-Vektoren
FDF9	FE49	Parameter für Filenamen setzen
FE00	FE50	Parameter für aktives File setzen
FE07	FE57	Status holen
FE18	FE66	Flag für Meldungen des Betriebssystems setzen
FE1C	FE6A	Status setzen
FE21	FEF6	Timeout-Flag für IEC-Bus setzen
FE25	FE73	RAM-Obergrenze setzen bzw. holen
FE34	FE82	RAM-Untergrenze setzen bzw. holen
FE43	FEA9	NMI-Routine
FEC2	FF5C	Konstanten für RS 232 Baud-Rate
FF48	FF72	Interrupthandler
FF81	FF8A	Sprungtabelle der Betriebssystem-Routinen

Nützliche Adressen des VC 20 Betriebssystems

Wenn man eigene Programme in Maschinensprache schreibt, kann man sich durch geschickte Ausnutzung der ROM-Routinen viel Arbeit sparen. Besonders die Routinen zur Bedienung der Peripheriegeräte bieten sich dazu an.

Die wichtigsten Routinen des VC 20 sind am Ende des ROMs als Sprungtabelle auf die eigentlichen Routinen zusammengefaßt. Diese Adressen wurden beim Erscheinen vom neuen Commodore Rechnern nicht geändert, sondern nur erweitert. Deshalb es ist zum Beispiel möglich Routinen, die für einen großen CBM-Rechner oder den Commodore 64 geschrieben wurden, ohne Schwierigkeiten auf dem VC 20 zu übernehmen, sofern nur diese sogenannten 'Kernal'-Routinen benutzt wurden. Die Sprungtabelle des VC 20 ist mit der des Commodore 64 identisch (dieser enthält nur noch drei zusätzliche Adressen), sodaß es mit Hilfe dieser Routinen ein leichtes ist, Programme des VC 20 zu konvertieren. Wir wollen uns jetzt einige dieser Routinen etwas näher ansehen.

Adresse	Funktion
$FF90	setzt Flag für Ausgabe von Systemmeldungen
$FF93	schickt Sekundäradresse nach einem LISTEN-Befehl auf den IEC-Bus
$FF96	schickt Sekundäradresse nach einem TALK-Befehl auf den IEC-Bus
$FF99	holt bei gesetzem Carry-Flag die höchste RAM-Adresse nach X und Y, bei gelöschtem Carry-Flag wird die Adresse von X und Y gesetzt.
$FF9C	die selbe Funktion wie oben, jedoch für den RAM-Anfang
$FF9F	frägt die Tastatur ab
$FFA2	setzt das Time-out Flag für den IEC-Bus
$FFA5	holt ein Byte vom IEC-Bus in den Akku
$FFA8	gibt ein Byte aus dem Akku an den IEC-Bus aus
$FFAB	sendet UNTALK-Befehl auf den IEC-Bus
$FFAE	sendet UNLISTEN-Befehl auf den IEC-Bus
$FFB1	sendet LISTEN-Befehl auf den IEC-Bus
$FFB4	sendet TALK-Befehl zum IEC-Bus
$FFB7	holt das Statuswort in den Akku
$FFBA	setzt die Fileparameter, Akku muß logischen Filenummer enthalten, X = Gerätenummer und Y = Sekundäradresse
$FFBD	setzt Parameter des Filenamens, Akku muß Länge des Namens enthalten, X und Y enthalten

Adresse	Funktion
	die Adresse des Filenamens
$FFC0	OPEN-Befehl, öffnet logische Datei
$FFC3	CLOSE-Befehl, schließt logische Datei, Akku muß logische Filenummer enthalten
$FFC6	CHKIN setzt folgende Eingabe auf logische Datei, die in X übergeben wird Die logische Datei muß vorher mit der OPEN-Routine geöffnet werden
$FFC9	CKOUT setzt folgende Ausgabe auf logische Datei, die in X übergeben wird Die logische Datei muß vorher mit der OPEN-Routine geöffnet werden
$FFCC	CLRCH setzt die Ein- und Ausgabe wieder auf Standard (Tastatur/Bildschirm)
$FFCF	BASIN Eingabe, holt ein Zeichen in den Akku
$FFD2	BSOUT Ausgabe, gibt Zeichen im Akku aus
$FFD5	LOAD, lädt Programm in den Speicher
$FFD8	SAVE, speichert Programm ab
$FFDB	setzt die laufende Zeit neu
$FFDE	holt die laufende Zeit
$FFE1	frägt die STOP-Taste ab
$FFE4	GET, holt ein Zeichen in den Akku
$FFE7	CLALL, setzt alle Ein-/Ausgabekanäle zurück, die Dateien werden jedoch nicht geschlossen
$FFEA	erhöht die laufende Zeit um eine sechzigstel Sekunde
$FFED	SCREEN holt die Anzahl der Zeilen und und Spalten des Bildschirms
$FFF0	bei gelöschtem Carry-Flag wird der Cursor auf die Position X/Y gesetzt, bei gesetztem Carry-Flag wird die Cursorposition geholt
$FFF3	holt die Startadresse des I/O-Bausteins

Zur Bedienung des Bildschirm stehen auch einige Routinen zur Verfügung, die Sie als Anwender benutzen können.

$E518	kompletter Reset des Bildschirms und der Tastaturabfrage

Adresse	Funktion
$E55F	CLR, löscht den Bildschirm
$E581	HOME, bringt den Cursor in die linke obere Ecke des Bildschirms
$E587	berechnet die Cursorposition
$E5BB	lädt den Videocontroller mit den Standardwerten
$E5CF	holt ein Zeichen aus dem Tastaturpuffer
$E5E5	wartet auf Tastatureingabe
$E975	Bildschirm scrollen, schiebt Bildschirm um eine Zeile nach oben
$EA8D	löscht eine Bildschirmzeile
$EAA1	setzt ein Zeichen mit Farbe auf dem Bildschirm (Bildschirmkode im Akku, Farbe in X)

```
C000  78 E3                      $E378   BASIC-Kaltstart
C002  67 E4                      $E467   BASIC-Warmstart
C004  43 42 4D 42 41 53 49 43    cbmbasic

******************************   Adressen der BASIC-Befehle (minus 1)
C00C  30 C8 41 C7
C010  1D CD F7 C8 A4 CB BE CB
C018  80 D0 05 CC A4 C9 9F C8
C020  70 C8 27 C9 1C C8 82 C8
C028  D1 C8 3A C9 2E C8 4A C9
C030  2C D8 64 E1 52 E1 61 E1
C038  B2 D3 23 D8 7F CA 8F CA
C040  56 C8 9B C6 5D C6 85 CA
C048  26 E1 BA E1 C3 E1 7A CB
C050  41 C6

******************************   Adressen der BASIC-Funktionen
C052  39 DC CC DC 58 DC
C058  00 00 7D D3 9E D3 71 DF
C060  94 E0 EA D9 ED DF 61 E2
C068  68 E2 B1 E2 0B E3 0D D8
C070  7C D7 65 D4 AD D7 8B D7
C078  EC D6 00 D7 2C D7 37 D7

******************************   Hierarchiekodes und Adressen
C080  79 69 D8 79 52 D8 7B 2A    der BASIC-Operatoren
C088  DA 7B 11 DB 7F 7A DF 50
C090  E8 CF 46 E5 CF 7D B3 DF
C098  5A D3 CE 64 15 D0

******************************   BASIC-Befehlsworte
C09E  45 4E                      en
C0A0  C4 46 4F D2 4E 45 58 D4    DfoRnexT
C0A8  44 41 54 C1 49 4E 50 55    datAinpu
C0B0  54 A3 49 4E 50 55 D4 44    t#inpuTd
C0B8  49 CD 52 45 41 C4 4C 45    iMreaDle
C0C0  D4 47 4F 54 CF 52 55 CE    TgotOruN
C0C8  49 C6 52 45 53 54 4F 52    iFrestor
C0D0  C5 47 4F 53 55 C2 52 45    EgosuBre
C0D8  54 55 52 CE 52 45 CD 53    turNreMs
C0E0  54 4F D0 4F CE 57 41 49    toPoNwai
C0E8  D4 4C 4F 41 C4 53 41 56    TloaDsav
C0F0  C5 56 45 52 49 46 D9 44    EverifYd
C0F8  45 C6 50 4F 4B C5 50 52    eFpokEpr
C100  49 4E 54 A3 50 52 49 4E    int#prin
C108  D4 43 4F 4E D4 4C 49 53    TconTlis
C110  D4 43 4C D2 43 4D C4 53    TclRcmDs
C118  59 D3 4F 50 45 CE 43 4C    ySopeNcl
C120  4F 53 C5 47 45 D4 4E 45    osEgeTne
C128  D7 54 41 42 A8 54 CF 46    Wtab(tOf
C130  CE 53 50 43 A8 54 48 45    Nspc(the
C138  CE 4E 4F D4 53 54 45 D0    NnoTsteP
C140  AB AD AA AF DE 41 4E C4    +-*/^anD
C148  4F D2 BE BD BC 53 47 CE    oR<=>sgN
C150  49 4E D4 41 42 D3 55 53    inTabSus
C158  D2 46 52 C5 50 4F D3 53    RfrEpoSs
C160  51 D2 52 4E C4 4C 4F C7    qRrnDloG
```

```
C168   45 58 D0 43 4F D3 53 49    exPcoSsi
C170   CE 54 41 CE 41 54 CE 50    NtaNatNp
C178   45 45 CB 4C 45 CE 53 54    eeKleNst
C180   52 A4 56 41 CC 41 53 C3    r$vaLasC
C188   43 48 52 A4 4C 45 46 54    chr$left
C190   A4 52 49 47 48 54 A4 4D    $right$m
C198   49 44 A4 47 CF 00          od$gO

****************************      BASIC-Fehlermeldungen
C19E   54 4F                      to
C1A0   4F 20 4D 41 4E 59 20 46    o many f
C1A8   49 4C 45 D3 46 49 4C 45    ileSfile
C1B0   20 4F 50 45 CE 46 49 4C     opeNfil
C1B8   45 20 4E 4F 54 20 4F 50    e not op
C1C0   45 CE 46 49 4C 45 20 4E    eNfile n
C1C8   4F 54 20 46 4F 55 4E C4    ot founD
C1D0   44 45 56 49 43 45 20 4E    device n
C1D8   4F 54 20 50 52 45 53 45    ot prese
C1E0   4E D4 4E 4F 54 20 49 4E    nTnot in
C1E8   50 55 54 20 46 49 4C C5    put filE
C1F0   4E 4F 54 20 4F 55 54 50    not outp
C1F8   55 54 20 46 49 4C C5 4D    ut filEm
C200   49 53 53 49 4E 47 20 46    issing f
C208   49 4C 45 20 4E 41 4D C5    ile namE
C210   49 4C 4C 45 47 41 4C 20    illegal 
C218   44 45 56 49 43 45 20 4E    device n
C220   55 4D 42 45 D2 4E 45 58    umbeRnex
C228   54 20 57 49 54 48 4F 55    t withou
C230   54 20 46 4F D2 53 59 4E    t foRsyn
C238   54 41 D8 52 45 54 55 52    taXretur
C240   4E 20 57 49 54 48 4F 55    n withou
C248   54 20 47 4F 53 55 C2 4F    t gosuBo
C250   55 54 20 4F 46 20 44 41    ut of da
C258   54 C1 49 4C 4C 45 47 41    tAillega
C260   4C 20 51 55 41 4E 54 49    l quanti
C268   54 D9 4F 56 45 52 46 4C    tYoverfl
C270   4F D7 4F 55 54 20 4F 46    oWout of
C278   20 4D 45 4D 4F 52 D9 55     memorYu
C280   4E 44 45 46 27 44 20 53    ndef'd s
C288   54 41 54 45 4D 45 4E D4    tatemenT
C290   42 41 44 20 53 55 42 53    bad subs
C298   43 52 49 50 D4 52 45 44    cripTred
C2A0   49 4D 27 44 20 41 52 52    im'd arr
C2A8   41 D9 44 49 56 49 53 49    aYdivisi
C2B0   4F 4E 20 42 59 20 5A 45    on by ze
C2B8   52 CF 49 4C 4C 45 47 41    rOillega
C2C0   4C 20 44 49 52 45 43 D4    l direcT
C2C8   54 59 50 45 20 4D 49 53    type mis
C2D0   4D 41 54 43 C8 53 54 52    matcHstr
C2D8   49 4E 47 20 54 4F 4F 20    ing too 
C2E0   4C 4F 4E C7 46 49 4C 45    lonGfile
C2E8   20 44 41 54 C1 46 4F 52     datAfor
C2F0   4D 55 4C 41 20 54 4F 4F    mula too
C2F8   20 43 4F 4D 50 4C 45 D8     compleX
C300   43 41 4E 27 54 20 43 4F    can't co
C308   4E 54 49 4E 55 C5 55 4E    ntinuEun
C310   44 45 46 27 44 20 46 55    def'd fu
C318   4E 43 54 49 4F CE 56 45    nctioNve
C320   52 49 46 D9 4C 4F 41 C4    rifYloaD
```

```
*****************************   Adressen der Fehlermeldungen
C328  9E C1 AC C1 B5 C1 C2 C1
C330  D0 C1 E2 C1 F0 C1 FF C1
C338  10 C2 25 C2 35 C2 3B C2
C340  4F C2 5A C2 6A C2 72 C2
C348  7F C2 90 C2 9D C2 AA C2
C350  BA C2 C8 C2 D5 C2 E4 C2
C358  ED C2 00 C3 0E C3 1E C3
C360  24 C3 83 C3

*****************************   Meldungen des Interpreters
C364  0D 4F 4B 0D               .ok.
C368  00 0D 20 45 52 52 4F 52   .. error
C370  00 20 49 4E 20 00 0D 0A   . in ...
C378  52 45 41 44 59 2E 0D 0A   ready...
C380  00 0D 0A 42 52 45 41 4B   ...break
C388  00 A0

*****************************   Stapelsuchroutine für FOR-NEXT + GOSUB
C38A  BA          TSX
C38B  E8          INX
C38C  E8          INX
C38D  E8          INX
C38E  E8          INX
C38F  BD 01 01    LDA $0101,X
C392  C9 81       CMP #$81      FOR-Kode
C394  D0 21       BNE $C3B7
C396  A5 4A       LDA $4A
C398  D0 0A       BNE $C3A4
C39A  BD 02 01    LDA $0102,X
C39D  85 49       STA $49       Variablenname vergleichen
C39F  BD 03 01    LDA $0103,X
C3A2  85 4A       STA $4A
C3A4  DD 03 01    CMP $0103,X
C3A7  D0 07       BNE $C3B0
C3A9  A5 49       LDA $49
C3AB  DD 02 01    CMP $0102,X
C3AE  F0 07       BEQ $C3B7
C3B0  8A          TXA
C3B1  18          CLC
C3B2  69 12       ADC #$12      18 Bytes für nächste Schleife addieren
C3B4  AA          TAX
C3B5  D0 D8       BNE $C38F
C3B7  60          RTS

*****************************   Blocktransfer-Routine
C3B8  20 08 C4    JSR $C408     prüft auf Platz im Speicher
C3BB  85 31       STA $31         Eingabe:
C3BD  84 32       STY $32
C3BF  38          SEC             $5F/$60  Alter Blockanfang
C3C0  A5 5A       LDA $5A         $5A/$5B  Altes Blockende + 1
C3C2  E5 5F       SBC $5F         $58/$59  Neues Blockende + 1
C3C4  85 22       STA $22
C3C6  A8          TAY             Einsprung  $C3BF
C3C7  A5 5B       LDA $5B
C3C9  E5 60       SBC $60
C3CB  AA          TAX
C3CC  E8          INX
C3CD  98          TYA
C3CE  F0 23       BEQ $C3F3
```

```
C3D0  A5 5A       LDA $5A
C3D2  38          SEC
C3D3  E5 22       SBC $22
C3D5  85 5A       STA $5A
C3D7  B0 03       BCS $C3DC
C3D9  C6 5B       DEC $5B
C3DB  38          SEC
C3DC  A5 58       LDA $58
C3DE  E5 22       SBC $22
C3E0  85 58       STA $58
C3E2  B0 08       BCS $C3EC
C3E4  C6 59       DEC $59
C3E6  90 04       BCC $C3EC
C3E8  B1 5A       LDA ($5A),Y
C3EA  91 58       STA ($58),Y
C3EC  88          DEY
C3ED  D0 F9       BNE $C3E8
C3EF  B1 5A       LDA ($5A),Y
C3F1  91 58       STA ($58),Y
C3F3  C6 5B       DEC $5B
C3F5  C6 59       DEC $59
C3F7  CA          DEX
C3F8  D0 F2       BNE $C3EC
C3FA  60          RTS

*****************************   Platz im Stapel ?
C3FB  0A          ASL           Akku muß dazu die halbe Zahl an
C3FC  69 3E       ADC #$3E      erforderlichen Platz enthalten
C3FE  B0 35       BCS $C435     gibt 'out of memory'
C400  85 22       STA $22
C402  BA          TSX
C403  E4 22       CPX $22
C405  90 2E       BCC $C435
C407  60          RTS

*****************************   Platz im Speicher ?
C408  C4 34       CPY $34
C40A  90 28       BCC $C434      A/Y  Adresse bis zu der Platz
C40C  D0 04       BNE $C412      benötigt wird.
C40E  C5 33       CMP $33
C410  90 22       BCC $C434
C412  48          PHA
C413  A2 09       LDX #$09
C415  98          TYA
C416  48          PHA
C417  B5 57       LDA $57,X     Register für Arithmetik retten
C419  CA          DEX
C41A  10 FA       BPL $C416
C41C  20 26 D5    JSR $D526     Garbage Collection
C41F  A2 F7       LDX #$F7
C421  68          PLA
C422  95 61       STA $61,X     Register zurückholen
C424  E8          INX
C425  30 FA       BMI $C421
C427  68          PLA
C428  A8          TAY
C429  68          PLA
C42A  C4 34       CPY $34
C42C  90 06       BCC $C434     Ok, fertig
C42E  D0 05       BNE $C435     kein Platz, dann 'out of memory'
```

```
C430  C5 33       CMP $33
C432  B0 01       BCS $C435
C434  60          RTS

C435  A2 10       LDX #$10      Fehlernummer für 'out of memory'

*****************************   Fehlermeldung ausgeben
C437  6C 00 03    JMP ($0300)   JMP $C43A
C43A  8A          TXA           XR enthält Fehlernummer $01..$1E
C43B  0A          ASL
C43C  AA          TAX
C43D  BD 26 C3    LDA $C326,X
C440  85 22       STA $22       Adresse der Fehlermeldung holen
C442  BD 27 C3    LDA $C327,X
C445  85 23       STA $23
C447  20 CC FF    JSR $FFCC     CLRCH  I/O-Kanal rücksetzen
C44A  A9 00       LDA #$00
C44C  85 13       STA $13       Ausgabeflag rücksetzen
C44E  20 D7 CA    JSR $CAD7     'CR' und 'LF' ausgeben
C451  20 45 CB    JSR $CB45     '?' ausgeben
C454  A0 00       LDY #$00
C456  B1 22       LDA ($22),Y   Text der Fehlermeldung
C458  48          PHA
C459  29 7F       AND #$7F
C45B  20 47 CB    JSR $CB47     ausgeben
C45E  C8          INY
C45F  68          PLA
C460  10 F4       BPL $C456
C462  20 7A C6    JSR $C67A     BASIC-Zeiger initialisieren, CONT sperren
C465  A9 69       LDA #$69
C467  A0 C3       LDY #$C3      Zeiger auf 'error'
C469  20 1E CB    JSR $CB1E     String ausgeben
C46C  A4 3A       LDY $3A
C46E  C8          INY
C46F  F0 03       BEQ $C474     Direkt-Modus ?
C471  20 C2 DD    JSR $DDC2     'in' Zeilennummer ausgeben
C474  A9 76       LDA #$76
C476  A0 C3       LDY #$C3      Zeiger auf 'ready.'
C478  20 1E CB    JSR $CB1E     String ausgeben
C47B  A9 80       LDA #$80
C47D  20 90 FF    JSR $FF90     Flag für Direkt-Modus setzen

*****************************   Eingabe-Warteschleife
C480  6C 02 03    JMP ($0302)   JMP $C483
C483  20 60 C5    JSR $C560     BASIC-Zeile in Eingabepuffer holen
C486  86 7A       STX $7A
C488  84 7B       STY $7B       CHRGET-Pointer auf Eingabepuffer
C48A  20 73 00    JSR $0073     CHRGET  nächstes Zeichen holen
C48D  AA          TAX
C48E  F0 F0       BEQ $C480     Puffer leer, dann weiter warten
C490  A2 FF       LDX #$FF
C492  86 3A       STX $3A       Kennzeichen für Direkt-Modus
C494  90 06       BCC $C49C     Ziffer, dann Programmzeile einfügen
C496  20 79 C5    JSR $C579     BASIC-Zeile in Interpreter-Code wandeln
C499  4C E1 C7    JMP $C7E1     Befehl ausführen

*****************************   Löschen + Einfügen von Programmzeilen
C49C  20 6B C9    JSR $C96B     Zeilennummer holen
C49F  20 79 C5    JSR $C579     BASIC-Zeile in Interpreter-Code wandeln
C4A2  84 0B       STY $0B
```

```
C4A4   20 13 C6    JSR $C613     Zeilenadresse holen
C4A7   90 44       BCC $C4ED     vorhanden? nein, dann Löschen übergehen
C4A9   A0 01       LDY #$01
C4AB   B1 5F       LDA ($5F),Y
C4AD   85 23       STA $23
C4AF   A5 2D       LDA $2D
C4B1   85 22       STA $22
C4B3   A5 60       LDA $60
C4B5   85 25       STA $25
C4B7   A5 5F       LDA $5F
C4B9   88          DEY
C4BA   F1 5F       SBC ($5F),Y
C4BC   18          CLC
C4BD   65 2D       ADC $2D
C4BF   85 2D       STA $2D
C4C1   85 24       STA $24
C4C3   A5 2E       LDA $2E
C4C5   69 FF       ADC #$FF
C4C7   85 2E       STA $2E
C4C9   E5 60       SBC $60
C4CB   AA          TAX
C4CC   38          SEC
C4CD   A5 5F       LDA $5F
C4CF   E5 2D       SBC $2D
C4D1   A8          TAY
C4D2   B0 03       BCS $C4D7
C4D4   E8          INX
C4D5   C6 25       DEC $25
C4D7   18          CLC
C4D8   65 22       ADC $22
C4DA   90 03       BCC $C4DF
C4DC   C6 23       DEC $23
C4DE   18          CLC
C4DF   B1 22       LDA ($22),Y   Verschiebeschleife
C4E1   91 24       STA ($24),Y
C4E3   C8          INY
C4E4   D0 F9       BNE $C4DF
C4E6   E6 23       INC $23
C4EB   E6 25       INC $25
C4EA   CA          DEX
C4EB   D0 F2       BNE $C4DF

******************************   Programmzeile einfügen
C4ED   20 59 C6    JSR $C659     CLR-Befehl
C4F0   20 33 C5    JSR $C533     Programmzeilen neu binden
C4F3   AD 00 02    LDA $0200     Zeichen im Puffer ?
C4F6   F0 88       BEQ $C480     nein, dann zur Warteschleife
C4F8   18          CLC
C4F9   A5 2D       LDA $2D
C4FB   85 5A       STA $5A
C4FD   65 0B       ADC $0B
C4FF   85 58       STA $58
C501   A4 2E       LDY $2E
C503   84 5B       STY $5B
C505   90 01       BCC $C508
C507   C8          INY
C508   84 59       STY $59
C50A   20 B8 C3    JSR $C3B8     BASIC-Zeilen verschieben
C50D   A5 14       LDA $14
C50F   A4 15       LDY $15
```

```
C511  8D FE 01    STA $01FE
C514  8C FF 01    STY $01FF
C517  A5 31       LDA $31
C519  A4 32       LDY $32
C51B  85 2D       STA $2D
C51D  84 2E       STY $2E
C51F  A4 0B       LDY $0B
C521  88          DEY
C522  B9 FC 01    LDA $01FC,Y
C525  91 5F       STA ($5F),Y
C527  88          DEY
C528  10 F8       BPL $C522
C52A  20 59 C6    JSR $C659     CLR-Befehl
C52D  20 33 C5    JSR $C533     Programmzeilen neu binden
C530  4C 80 C4    JMP $C480     zur Eingabe-Warteschleife

******************************  Programmzeilen neu binden
C533  A5 2B       LDA $2B
C535  A4 2C       LDY $2C
C537  85 22       STA $22
C539  84 23       STY $23
C53B  18          CLC
C53C  A0 01       LDY #$01
C53E  B1 22       LDA ($22),Y
C540  F0 1D       BEQ $C55F
C542  A0 04       LDY #$04
C544  C8          INY
C545  B1 22       LDA ($22),Y
C547  D0 FB       BNE $C544
C549  C8          INY
C54A  98          TYA
C54B  65 22       ADC $22
C54D  AA          TAX
C54E  A0 00       LDY #$00
C550  91 22       STA ($22),Y
C552  A5 23       LDA $23
C554  69 00       ADC #$00
C556  C8          INY
C557  91 22       STA ($22),Y
C559  86 22       STX $22
C55B  85 23       STA $23
C55D  90 DD       BCC $C53C
C55F  60          RTS

******************************  Eingabe einer Zeile
C560  A2 00       LDX #$00
C562  20 0F E1    JSR $E10F     BASIN  ein Zeichen holen
C565  C9 0D       CMP #$0D      RETURN-Taste ?
C567  F0 0D       BEQ $C576     ja, dann Eingabe beenden
C569  9D 00 02    STA $0200,X   Zeichen im Eingabepuffer speichern
C56C  E8          INX
C56D  E0 59       CPX #$59      Eingabepuffer voll ?
C56F  90 F1       BCC $C562
C571  A2 17       LDX #$17      Nummer für 'string too long'
C573  4C 37 C4    JMP $C437     Fehlermeldung ausgeben
C576  4C CA CA    JMP $CACA     Puffer mit 0 abschließen, CR ausgeben

******************************  Umwandl. einer Zeile in Interpreter-Kode
C579  6C 04 03    JMP ($0304)   JMP $C57C
C57C  A6 7A       LDX $7A
```

```
C57E  A0 04     LDY #$04
C580  84 0F     STY $0F       Hochkomma-Flag
C582  BD 00 02  LDA $0200,X   Zeichen aus Puffer holen
C585  10 07     BPL $C58E     kein Interpreter-Kode ?
C587  C9 FF     CMP #$FF      Kode für Pi
C589  F0 3E     BEQ $C5C9
C58B  E8        INX
C58C  D0 F4     BNE $C582
C58E  C9 20     CMP #$20      ' ' Leerzeichen
C590  F0 37     BEQ $C5C9
C592  85 08     STA $08
C594  C9 22     CMP #$22      '"' Hochkomma
C596  F0 56     BEQ $C5EE
C598  24 0F     BIT $0F
C59A  70 2D     BVS $C5C9
C59C  C9 3F     CMP #$3F      '?' Fragezeichen
C59E  D0 04     BNE $C5A4
C5A0  A9 99     LDA #$99      durch Kode für PRINT ersetzen
C5A2  D0 25     BNE $C5C9
C5A4  C9 30     CMP #$30      '0'
C5A6  90 04     BCC $C5AC
C5A8  C9 3C     CMP #$3C
C5AA  90 1D     BCC $C5C9
C5AC  84 71     STY $71
C5AE  A0 00     LDY #$00
C5B0  84 0B     STY $0B
C5B2  88        DEY
C5B3  86 7A     STX $7A
C5B5  CA        DEX
C5B6  C8        INY
C5B7  E8        INX
C5B8  BD 00 02  LDA $0200,X   Zeichen im Puffer
C5BB  38        SEC
C5BC  F9 9E C0  SBC $C09E,Y   mit Befehlsworten in Tabelle vergleichen
C5BF  F0 F5     BEQ $C5B6
C5C1  C9 80     CMP #$80
C5C3  D0 30     BNE $C5F5
C5C5  05 0B     ORA $0B       gefunden, Interpreter-Kode gleich Zähler +$80
C5C7  A4 71     LDY $71
C5C9  E8        INX
C5CA  C8        INY
C5CB  99 FB 01  STA $01FB,Y   Interpreterkode speichern
C5CE  B9 FB 01  LDA $01FB,Y   und Statusregister setzen
C5D1  F0 36     BEQ $C609     Ende, dann fertig
C5D3  38        SEC
C5D4  E9 3A     SBC #$3A      ':'  Trennzeichen
C5D6  F0 04     BEQ $C5DC     ja
C5D8  C9 49     CMP #$49      DATA - Kode ?
C5DA  D0 02     BNE $C5DE
C5DC  85 0F     STA $0F
C5DE  38        SEC
C5DF  E9 55     SBC #$55      REM-Kode ?
C5E1  D0 9F     BNE $C582
C5E3  85 08     STA $08
C5E5  BD 00 02  LDA $0200,X
C5E8  F0 DF     BEQ $C5C9
C5EA  C5 08     CMP $08
C5EC  F0 DB     BEQ $C5C9
C5EE  C8        INY
C5EF  99 FB 01  STA $01FB,Y
```

```
C5F2  E8          INX
C5F3  D0 F0       BNE $C5E5
C5F5  A6 7A       LDX $7A
C5F7  E6 0B       INC $0B
C5F9  C8          INY
C5FA  B9 9D C0    LDA $C09D,Y
C5FD  10 FA       BPL $C5F9
C5FF  B9 9E C0    LDA $C09E,Y    mit Tabelle vergleichen
C602  D0 B4       BNE $C5B8
C604  BD 00 02    LDA $0200,X
C607  10 BE       BPL $C5C7
C609  99 FD 01    STA $01FD,Y
C60C  C6 7B       DEC $7B
C60E  A9 FF       LDA #$FF       Programmzeiger auf Eingabepuffer - 1
C610  85 7A       STA $7A
C612  60          RTS

*****************************    Startadr. einer Prog.-Zeile berechnen
C613  A5 2B       LDA $2B
C615  A6 2C       LDX $2C
C617  A0 01       LDY #$01       Zeilennummer in 14/15
C619  85 5F       STA $5F        Adresse in 5F/60
C61B  86 60       STX $60
C61D  B1 5F       LDA ($5F),Y    Zeile vorhanden dann C=1
C61F  F0 1F       BEQ $C640      Zeile nicht vorhanden dann C=0
C621  C8          INY            $5F/$60 enthält dann Adresse
C622  C8          INY            der nächsten Zeile
C623  A5 15       LDA $15
C625  D1 5F       CMP ($5F),Y
C627  90 18       BCC $C641
C629  F0 03       BEQ $C62E
C62B  88          DEY
C62C  D0 09       BNE $C637
C62E  A5 14       LDA $14
C630  88          DEY
C631  D1 5F       CMP ($5F),Y
C633  90 0C       BCC $C641
C635  F0 0A       BEQ $C641
C637  88          DEY
C638  B1 5F       LDA ($5F),Y
C63A  AA          TAX
C63B  88          DEY
C63C  B1 5F       LDA ($5F),Y
C63E  B0 D7       BCS $C617
C640  18          CLC
C641  60          RTS

*****************************    Basic-Befehl  NEW
C642  D0 FD       BNE $C641
C644  A9 00       LDA #$00
C646  A8          TAY
C647  91 2B       STA ($2B),Y
C649  C8          INY            Zweimal $00 an Programmstart
C64A  91 2B       STA ($2B),Y
C64C  A5 2B       LDA $2B
C64E  18          CLC
C64F  69 02       ADC #$02
C651  85 2D       STA $2D
C653  A5 2C       LDA $2C        Variablenstart = Programmstart + 2
C655  69 00       ADC #$00
```

```
C657  85 2E      STA $2E
C659  20 8E C6   JSR $C68E     CHRGET-Zeiger auf Programmstart
C65C  A9 00      LDA #$00

*****************************  Basic-Befehl   CLR
C65E  D0 2D      BNE $C68D
C660  20 E7 FF   JSR $FFE7     CLALL  I/O-Kanäle zurücksetzen
C663  A5 37      LDA $37
C665  A4 38      LDY $38
C667  85 33      STA $33       Stringstart auf BASIC-RAM-Ende
C669  84 34      STY $34
C66B  A5 2D      LDA $2D
C66D  A4 2E      LDY $2E
C66F  85 2F      STA $2F
C671  84 30      STY $30       Variablenende = Variablenanfang
C673  85 31      STA $31
C675  84 32      STY $32
C677  20 1D C8   JSR $C81D     RESTORE-Befehl
C67A  A2 19      LDX #$19
C67C  86 16      STX $16       Descriptor-Index rücksetzen
C67E  68         PLA
C67F  A8         TAY           Rücksprungadresse holen
C680  68         PLA
C681  A2 FA      LDX #$FA      Stackpointer initialisieren
C683  9A         TXS
C684  48         PHA
C685  98         TYA           Rücksprungadresse zurücksetzen
C686  48         PHA
C687  A9 00      LDA #$00
C689  85 3E      STA $3E       CONTINUE sperren
C68B  85 10      STA $10
C68D  60         RTS

*****************************  Programmzeiger auf BASIC-Start - 1
C68E  18         CLC
C68F  A5 2B      LDA $2B
C691  69 FF      ADC #$FF
C693  85 7A      STA $7A
C695  A5 2C      LDA $2C
C697  69 FF      ADC #$FF
C699  85 7B      STA $7B
C69B  60         RTS

*****************************  Basic-Befehl   LIST
C69C  90 06      BCC $C6A4     Ziffer (Zeilennummer) ?
C69E  F0 04      BEQ $C6A4     nur LIST
C6A0  C9 AB      CMP #$AB      '-' Kode
C6A2  D0 E9      BNE $C68D
C6A4  20 6B C9   JSR $C96B     Zeilennummer holen
C6A7  20 13 C6   JSR $C613     Zeilenadresse holen
C6AA  20 79 00   JSR $0079     CHRGOT  laufendes Zeichen holen
C6AD  F0 0C      BEQ $C6BB
C6AF  C9 AB      CMP #$AB      '-' Kode
C6B1  D0 8E      BNE $C641     nein, dann  SYNTAX ERROR
C6B3  20 73 00   JSR $0073     CHRGET  nächstes Zeichen holen
C6B6  20 6B C9   JSR $C96B     Zeilennummer holen
C6B9  D0 86      BNE $C641
C6BB  68         PLA
C6BC  68         PLA
C6BD  A5 14      LDA $14
```

```
C6BF   05 15       ORA $15        zweite Zeilennummer gleich null ?
C6C1   D0 06       BNE $C6C9
C6C3   A9 FF       LDA #$FF
C6C5   85 14       STA $14
C6C7   85 15       STA $15
C6C9   A0 01       LDY #$01
C6CB   84 0F       STY $0F
C6CD   B1 5F       LDA ($5F),Y    Linkadresse high
C6CF   F0 43       BEQ $C714      Programmende, dann fertig
C6D1   20 2C C8    JSR $C82C      prüft auf STOP-Taste
C6D4   20 D7 CA    JSR $CAD7      'CR' und 'LF' ausgeben
C6D7   C8          INY
C6D8   B1 5F       LDA ($5F),Y
C6DA   AA          TAX
C6DB   C8          INY
C6DC   B1 5F       LDA ($5F),Y
C6DE   C5 15       CMP $15
C6E0   D0 04       BNE $C6E6
C6E2   E4 14       CPX $14
C6E4   F0 02       BEQ $C6E8
C6E6   B0 2C       BCS $C714
C6E8   84 49       STY $49
C6EA   20 CD DD    JSR $DDCD      Zeilennumer ausgeben
C6ED   A9 20       LDA #$20       ' ' Leerzeichen
C6EF   A4 49       LDY $49
C6F1   29 7F       AND #$7F
C6F3   20 47 CB    JSR $CB47      Zeichen ausgeben
C6F6   C9 22       CMP #$22       '"' Hochkomma
C6F8   D0 06       BNE $C700
C6FA   A5 0F       LDA $0F
C6FC   49 FF       EOR #$FF       Hochkommaflag umdrehen
C6FE   85 0F       STA $0F
C700   C8          INY
C701   F0 11       BEQ $C714
C703   B1 5F       LDA ($5F),Y
C705   D0 10       BNE $C717      kein Zeilenende, dann listen
C707   A8          TAY
C708   B1 5F       LDA ($5F),Y
C70A   AA          TAX
C70B   C8          INY
C70C   B1 5F       LDA ($5F),Y
C70E   86 5F       STX $5F
C710   85 60       STA $60
C712   D0 B5       BNE $C6C9
C714   4C 74 C4    JMP $C474      zum BASIC-Warmstart

*******************************   Interpreter-Kode in Klartext umwandeln
C717   6C 06 03    JMP ($0306)    JMP $C71A
C71A   10 D7       BPL $C6F3      kein Interpreterkode, so ausgeben
C71C   C9 FF       CMP #$FF       Kode für Pi
C71E   F0 D3       BEQ $C6F3      so ausgeben
C720   24 0F       BIT $0F        Hochkommamodus
C722   30 CF       BMI $C6F3      dann Zeichen so ausgeben
C724   38          SEC
C725   E9 7F       SBC #$7F
C727   AA          TAX
C728   84 49       STY $49
C72A   A0 FF       LDY #$FF
C72C   CA          DEX
C72D   F0 08       BEQ $C737      erstes Befehlswort ?
```

```
C72F  C8        INY
C730  B9 9E C0  LDA $C09E,Y   Offset für Xtes Befehlswort finden
C733  10 FA     BPL $C72F
C735  30 F5     BMI $C72C     Bit 7 gesetzt, nächstes Wort
C737  C8        INY
C738  B9 9E C0  LDA $C09E,Y   Befehlswort aus Tabelle holen
C73B  30 B2     BMI $C6EF
C73D  20 47 CB  JSR $CB47     Zeichen ausgeben
C740  D0 F5     BNE $C737

******************************  BASIC-Befehl  FOR
C742  A9 80     LDA #$80      Integervariablen sperren
C744  85 10     STA $10
C746  20 A5 C9  JSR $C9A5     LET-Befehl, setzt Variablenwert
C749  20 8A C3  JSR $C38A     sucht offene FOR-NEXT-Schleife mit gleicher
C74C  D0 05     BNE $C753                                       Variabler
C74E  8A        TXA
C74F  69 0F     ADC #$0F
C751  AA        TAX
C752  9A        TXS
C753  68        PLA
C754  68        PLA
C755  A9 09     LDA #$09
C757  20 FB C3  JSR $C3FB     prüft auf genügend Platz im Stack
C75A  20 06 C9  JSR $C906     sucht nächstes BASIC-Statement
C75D  18        CLC
C75E  98        TYA
C75F  65 7A     ADC $7A       Programmzeiger auf nächsten Befehl
C761  48        PHA           auf Stack speichern
C762  A5 7B     LDA $7B
C764  69 00     ADC #$00
C766  48        PHA
C767  A5 3A     LDA $3A
C769  48        PHA           laufende Zeilennummer auf Stack
C76A  A5 39     LDA $39
C76C  48        PHA
C76D  A9 A4     LDA #$A4      'TO' Kode
C76F  20 FF CE  JSR $CEFF     prüft auf Kode
C772  20 8D CD  JSR $CD8D     prüft auf numerische Variable
C775  20 8A CD  JSR $CD8A     FRMNUM holt numerischen Ausdruck
C778  A5 66     LDA $66
C77A  09 7F     ORA #$7F
C77C  25 62     AND $62
C77E  85 62     STA $62
C780  A9 8B     LDA #$8B      Rücksprungadresse merken
C782  A0 C7     LDY #$C7
C784  85 22     STA $22
C786  84 23     STY $23
C788  4C 43 CE  JMP $CE43     legt Schleifenendwert auf Stack
C78B  A9 BC     LDA #$BC
C78D  A0 D9     LDY #$D9      Zeiger auf Konstante 1
C78F  20 A2 DB  JSR $DBA2     als Default-STEP-Wert in FAC
C792  20 79 00  JSR $0079     CHRGOT   laufendes Zeichen holen
C795  C9 A9     CMP #$A9      'STEP' -Kode
C797  D0 06     BNE $C79F
C799  20 73 00  JSR $0073     CHRGET  nächstes Zeichen holen
C79C  20 8A CD  JSR $CD8A     FRMNUM
C79F  20 2B DC  JSR $DC2B     Vorzeichen von FAC holen
C7A2  20 38 CE  JSR $CE38     legt Vorzeichen und STEP-Wert auf Stack
C7A5  A5 4A     LDA $4A
```

```
C7A7  48        PHA            Variablenname und
C7A8  A5 49     LDA $49
C7AA  48        PHA
C7AB  A9 81     LDA #$81       FOR-Kode auf Stack
C7AD  48        PHA

******************************  Interpreter-Schleife
C7AE  20 2C C8  JSR $C82C      prüft auf STOP-Taste
C7B1  A5 7A     LDA $7A
C7B3  A4 7B     LDY $7B        Programmzeiger
C7B5  C0 02     CPY #$02       Direct-Modus ?
C7B7  EA        NOP
C7B8  F0 04     BEQ $C7BE      ja
C7BA  85 3D     STA $3D
C7BC  84 3E     STY $3E        als Zeiger für CONT merken
C7BE  A0 00     LDY #$00
C7C0  B1 7A     LDA ($7A),Y    laufendens Zeichen
C7C2  D0 43     BNE $C807      nicht Zeilenende ?
C7C4  A0 02     LDY #$02
C7C6  B1 7A     LDA ($7A),Y
C7C8  18        CLC            Flag für END setzen
C7C9  D0 03     BNE $C7CE      Programmende ?
C7CB  4C 4B C8  JMP $C84B      ja, dann END ausführen
C7CE  C8        INY
C7CF  B1 7A     LDA ($7A),Y
C7D1  85 39     STA $39
C7D3  C8        INY            laufende Zeilennummer merken
C7D4  B1 7A     LDA ($7A),Y
C7D6  85 3A     STA $3A
C7D8  98        TYA
C7D9  65 7A     ADC $7A        Programmzeiger auf nächste Zeile
C7DB  85 7A     STA $7A
C7DD  90 02     BCC $C7E1
C7DF  E6 7B     INC $7B
C7E1  6C 08 03  JMP ($0308)    JMP $C7E4
C7E4  20 73 00  JSR $0073      CHRGET  nächstes Zeichen holen
C7E7  20 ED C7  JSR $C7ED      Statement ausführen
C7EA  4C AE C7  JMP $C7AE      zurück zur Interpreterschleife

******************************  BASIC-Statement dekodieren
C7ED  F0 3C     BEQ $C82B      Zeilenende, dann fertig
C7EF  E9 80     SBC #$80
C7F1  90 11     BCC $C804      Interpreterkode
C7F3  C9 23     CMP #$23
C7F5  B0 17     BCS $C80E      Funktionskode oder GO TO
C7F7  0A        ASL
C7F8  A8        TAY
C7F9  B9 0D C0  LDA $C00D,Y
C7FC  48        PHA            Befehlsadresse aus Tabelle holen
C7FD  B9 0C C0  LDA $C00C,Y
C800  48        PHA
C801  4C 73 00  JMP $0073      nächstes Zeichen holen und Befehl ausführen

C804  4C A5 C9  JMP $C9A5      zum LET-Befehl

******************************  prüft auf Folgestatement
C807  C9 3A     CMP #$3A       ':'
C809  F0 D6     BEQ $C7E1
C80B  4C 08 CF  JMP $CF08      gibt 'SYNTAX ERROR'
```

```
*****************************   prüft auf 'GO' 'TO' Kode
C80E  C9 4B       CMP #$4B      'GO' - Kode (minus 80)
C810  D0 F9       BNE $C80B
C812  20 73 00    JSR $0073     CHRGET  nächstes Zeichen holen
C815  A9 A4       LDA #$A4      'TO' - Kode
C817  20 FF CE    JSR $CEFF     prüft auf Kode
C81A  4C A0 C8    JMP $C8A0     zum GOTO-Befehl

*****************************   BASIC-Befehl  RESTORE
C81D  38          SEC
C81E  A5 2B       LDA $2B
C820  E9 01       SBC #$01      Programmstart - 1
C822  A4 2C       LDY $2C
C824  B0 01       BCS $C827
C826  88          DEY
C827  85 41       STA $41       gleich DATA-Zeiger
C829  84 42       STY $42
C82B  60          RTS

*****************************   prüft auf STOP-Taste
C82C  20 E1 FF    JSR $FFE1     STOP-Taste abfragen

*****************************   BASIC-Befehl  STOP
C82F  B0 01       BCS $C832     C=1  Flag für STOP

*****************************   BASIC-Befehl  END
C831  18          CLC           C=0  Flag für END
C832  D0 3C       BNE $C870
C834  A5 7A       LDA $7A
C836  A4 7B       LDY $7B       Programmzeiger
C838  A6 3A       LDX $3A
C83A  E8          INX
C83B  F0 0C       BEQ $C849     Direct-Modus ?
C83D  85 3D       STA $3D
C83F  84 3E       STY $3E       Zeiger für CONT
C841  A5 39       LDA $39
C843  A4 3A       LDY $3A       Nummer der laufenden Zeile
C845  85 3B       STA $3B
C847  84 3C       STY $3C       als Zeilennummer für CONT merken
C849  68          PLA
C84A  68          PLA           Rücksprungadresse vom Stack holen
C84B  A9 81       LDA #$81
C84D  A0 C3       LDY #$C3      Zeiger auf 'break'
C84F  90 03       BCC $C854     END Flag gesetzt ?
C851  4C 69 C4    JMP $C469     nein, dann 'break in Zeilennummer'
C854  4C 74 C4    JMP $C474     zum BASIC-Warmstart

*****************************   BASIC-Befehl  CONT
C857  D0 17       BNE $C870
C859  A2 1A       LDX #$1A      Fehlernummer für 'can't continue'
C85B  A4 3E       LDY $3E       CONT gesperrt ?
C85D  D0 03       BNE $C862     nein
C85F  4C 37 C4    JMP $C437     Fehlermeldung ausgeben
C862  A5 3D       LDA $3D
C864  85 7A       STA $7A
C866  84 7B       STY $7B       Programmzeiger
C868  A5 3B       LDA $3B
C86A  A4 3C       LDY $3C       und
C86C  85 39       STA $39
C86E  84 3A       STY $3A       Zeilenummer setzen
```

```
C870  60          RTS

****************************** BASIC-Befehl  RUN
C871  08          PHP
C872  A9 00       LDA #$00
C874  20 90 FF    JSR $FF90     Flag für Programm-Modus setzen
C877  28          PLP
C878  D0 03       BNE $C87D     weitere Zeichen ?
C87A  4C 59 C6    JMP $C659     Programmzeiger auf Start, CLR
C87D  20 60 C6    JSR $C660     ja, CLR-Befehl
C880  4C 97 C8    JMP $C897     zum GOTO-Befehl

****************************** BASIC-Befehl  GOSUB
C883  A9 03       LDA #$03
C885  20 FB C3    JSR $C3FB     prüft auf genügend Platz im Stack
C888  A5 7B       LDA $7B
C88A  48          PHA           Programmzeiger
C88B  A5 7A       LDA $7A
C88D  48          PHA
C88E  A5 3A       LDA $3A
C890  48          PHA           Zeilennummer
C891  A5 39       LDA $39
C893  48          PHA
C894  A9 8D       LDA #$8D      und 'GOSUB'-Kode auf Stack
C896  48          PHA
C897  20 79 00    JSR $0079     CHRGOT  laufendes Zeichen holen
C89A  20 A0 C8    JSR $C8A0     GOTO-Befehl
C89D  4C AE C7    JMP $C7AE     zur Interpreterschleife

****************************** BASIC-Befehl  GOTO
C8A0  20 6B C9    JSR $C96B     Zeilennummer nach $14/$15 holen
C8A3  20 09 C9    JSR $C909     nächsten Zeilenanfang suchen
C8A6  38          SEC
C8A7  A5 39       LDA $39
C8A9  E5 14       SBC $14       ist Zeilennummer kleiner als laufende Zeile ?
C8AB  A5 3A       LDA $3A
C8AD  E5 15       SBC $15
C8AF  B0 0B       BCS $C8BC     nein
C8B1  98          TYA
C8B2  38          SEC
C8B3  65 7A       ADC $7A       sucht ab laufender Zeile
C8B5  A6 7B       LDX $7B
C8B7  90 07       BCC $C8C0
C8B9  E8          INX
C8BA  B0 04       BCS $C8C0
C8BC  A5 2B       LDA $2B       sucht ab Programmstart
C8BE  A6 2C       LDX $2C
C8C0  20 17 C6    JSR $C617     sucht Programmzeile
C8C3  90 1E       BCC $C8E3     nicht gefunden, 'undef'd statement'
C8C5  A5 5F       LDA $5F
C8C7  E9 01       SBC #$01
C8C9  85 7A       STA $7A       Programmzeiger auf neue Zeile setzen
C8CB  A5 60       LDA $60
C8CD  E9 00       SBC #$00
C8CF  85 7B       STA $7B
C8D1  60          RTS

****************************** BASIC-Befehl  RETURN
C8D2  D0 FD       BNE $C8D1
C8D4  A9 FF       LDA #$FF
```

```
C8D6  85 4A      STA $4A
C8D8  20 8A C3   JSR $C38A      nächsten GOSUB-Dataensatz im Stack suchen
C8DB  9A         TXS
C8DC  C9 8D      CMP #$8D       'GOSUB'-Kode
C8DE  F0 0B      BEQ $C8EB      gefunden ?
C8E0  A2 0C      LDX #$0C       Nummer für 'return without gosub'
C8E2  2C         .BYTE $2C
C8E3  A2 11      LDX #$11       Nummer für 'undef'd statement'
C8E5  4C 37 C4   JMP $C437      Fehlermeldung ausgeben
C8E8  4C 08 CF   JMP $CF08      'SYNTAX ERROR' ausgeben
C8EB  68         PLA
C8EC  68         PLA
C8ED  85 39      STA $39
C8EF  68         PLA            Zeilennummer
C8F0  85 3A      STA $3A
C8F2  68         PLA
C8F3  85 7A      STA $7A
C8F5  68         PLA            und Programmzeiger vom Stack holen
C8F6  85 7B      STA $7B

*****************************   BASIC-Befehl  DATA
C8F8  20 06 C9   JSR $C906      nächstes Statement suchen
C8FB  98         TYA
C8FC  18         CLC            Offset
C8FD  65 7A      ADC $7A
C8FF  85 7A      STA $7A        zum Programmzeiger addieren
C901  90 02      BCC $C905
C903  E6 7B      INC $7B
C905  60         RTS

*****************************   Offset des nächsten Trennzeichens finden
C906  A2 3A      LDX #$3A       ':'
C908  2C         .BYTE $2C
C909  A2 00      LDX #$00       0, Zeilenende
C90B  86 07      STX $07
C90D  A0 00      LDY #$00       Y enhält Offset
C90F  84 08      STY $08
C911  A5 08      LDA $08
C913  A6 07      LDX $07        gesuchtes Zeichen
C915  85 07      STA $07
C917  86 08      STX $08
C919  B1 7A      LDA ($7A),Y    Zeichen holen
C91B  F0 E8      BEQ $C905      Zeilenende, dann fertig
C91D  C5 08      CMP $08
C91F  F0 E4      BEQ $C905
C921  C8         INY            Zeiger erhöhen
C922  C9 22      CMP #$22       '"'  Hochkomma
C924  D0 F3      BNE $C919
C926  F0 E9      BEQ $C911

*****************************   BASIC-Befehl  IF
C928  20 9E CD   JSR $CD9E      FRMEVL  Ausdruck berechnen
C92B  20 79 00   JSR $0079      CHRGOT  laufendes Zeichen holen
C92E  C9 89      CMP #$89       'GOTO' - Kode
C930  F0 05      BEQ $C937      ja
C932  A9 A7      LDA #$A7       'THEN' - Kode
C934  20 FF CE   JSR $CEFF      prüft auf Kode
C937  A5 61      LDA $61
C939  D0 05      BNE $C940      Ausdruck wahr ?
```

```
*****************************   BASIC-Befehl  REM
C93B  20 09 C9   JSR $C909      nein, nächsten Zeilenanfang suchen
C93E  F0 BB      BEQ $C8FB      Programmzeiger auf nächste Zeile
C940  20 79 00   JSR $0079      CHRGOT  laufendes Zeichen holen
C943  B0 03      BCS $C948      keine Ziffer ?
C945  4C A0 C8   JMP $C8A0      zum GOTO-Befehl
C948  4C ED C7   JMP $C7ED      nächsten Befehl ausführen

*****************************   BASIC-Befehl  ON
C94B  20 9E D7   JSR $D79E      holt Byte-Wert (0-255)
C94E  48         PHA            Kode merken
C94F  C9 8D      CMP #$8D       'GOSUB' - Kode ?
C951  F0 04      BEQ $C957      ja
C953  C9 89      CMP #$89       'GOTO' - Kode ?
C955  D0 91      BNE $C8E8      nein, dann 'syntax error'
C957  C6 65      DEC $65        Zähler erniedrigen
C959  D0 04      BNE $C95F      noch nicht null ?
C95B  68         PLA            ja, Kode zurückholen
C95C  4C EF C7   JMP $C7EF      und Befehl ausführen

C95F  20 73 00   JSR $0073      CHRGET nächstes Zeichen holen
C962  20 6B C9   JSR $C96B      Zeilennummer holen
C965  C9 2C      CMP #$2C       ',' Komma ?
C967  F0 EE      BEQ $C957      ja, dann weiter
C969  68         PLA            kein Sprung, Kode zurück, fertig
C96A  60         RTS

*****************************   holt Zeilennummer und wandelt in Adressformat
C96B  A2 00      LDX #$00
C96D  86 14      STX $14
C96F  86 15      STX $15        Vorbesetzung für Zeilennummer gleich Null
C971  B0 F7      BCS $C96A      Ergebnis ist Nummer der Zeile in $14/$15
C973  E9 2F      SBC #$2F
C975  85 07      STA $07
C977  A5 15      LDA $15
C979  85 22      STA $22
C97B  C9 19      CMP #$19
C97D  B0 D4      BCS $C953
C97F  A5 14      LDA $14
C981  0A         ASL
C982  26 22      ROL $22
C984  0A         ASL
C985  26 22      ROL $22
C987  65 14      ADC $14
C989  85 14      STA $14
C98B  A5 22      LDA $22
C98D  65 15      ADC $15
C98F  85 15      STA $15
C991  06 14      ASL $14
C993  26 15      ROL $15
C995  A5 14      LDA $14
C997  65 07      ADC $07
C999  85 14      STA $14
C99B  90 02      BCC $C99F
C99D  E6 15      INC $15
C99F  20 73 00   JSR $0073      CHRGET  nächstes Zeichen holen
C9A2  4C 71 C9   JMP $C971      zur Auswertung

*****************************   BASIC-Befehl  LET
C9A5  20 8B D0   JSR $D08B      sucht Variable
```

```
C9A8  85 49        STA $49
C9AA  84 4A        STY $4A        Variablenadresse merken
C9AC  A9 B2        LDA #$B2       '=' - Kode
C9AE  20 FF CE     JSR $CEFF      prüft auf Kode
C9B1  A5 0E        LDA $0E        Integer-Flag
C9B3  48           PHA
C9B4  A5 0D        LDA $0D        String-Flag
C9B6  48           PHA
C9B7  20 9E CD     JSR $CD9E      FRMEVL  Ausdruck holen
C9BA  68           PLA            Typflag zurückholen
C9BB  2A           ROL
C9BC  20 90 CD     JSR $CD90      und auf gleichen Typ prüfen
C9BF  D0 18        BNE $C9D9
C9C1  68           PLA
C9C2  10 12        BPL $C9D6      Real ?

****************************      Wertzuweisung an Integer-Variable
C9C4  20 1B DC     JSR $DC1B      FAC runden
C9C7  20 BF D1     JSR $D1BF      und nach Integer wandeln
C9CA  A0 00        LDY #$00
C9CC  A5 64        LDA $64
C9CE  91 49        STA ($49),Y    Wert in Variable übertragen
C9D0  C8           INY
C9D1  A5 65        LDA $65
C9D3  91 49        STA ($49),Y
C9D5  60           RTS

****************************      Wertzuweisung an Real-Variable
C9D6  4C D0 DB     JMP $DBD0      FAC nach Variable bringen

****************************      Wertzuweisung an String-Variable
C9D9  68           PLA
C9DA  A4 4A        LDY $4A        Varibalenadresse high
C9DC  C0 DF        CPY #$DF       ist Varibale TI$ ?
C9DE  D0 4C        BNE $CA2C      nein
C9E0  20 A6 D6     JSR $D6A6      FRESTR
C9E3  C9 06        CMP #$06       Stringlänge gleich 6
C9E5  D0 3D        BNE $CA24      nein, dann 'illegal quantity'
C9E7  A0 00        LDY #$00
C9E9  84 61        STY $61
C9EB  84 66        STY $66
C9ED  84 71        STY $71
C9EF  20 1D CA     JSR $CA1D      prüft nächstes Zeichen auf Ziffer
C9F2  20 E2 DA     JSR $DAE2      FAC = FAC * 10
C9F5  E6 71        INC $71        Stellenzähler erhöhen
C9F7  A4 71        LDY $71
C9F9  20 1D CA     JSR $CA1D      prüft nächstes Zeichen auf Ziffer
C9FC  20 0C DC     JSR $DC0C      FAC nach ARG kopieren
C9FF  AA           TAX
CA00  F0 05        BEQ $CA07      FAC gleich null ?
CA02  E8           INX
CA03  8A           TXA
CA04  20 ED DA     JSR $DAED      FAC = FAC + ARG
CA07  A4 71        LDY $71        Stellenzähler
CA09  C8           INY            erhöhen
CA0A  C0 06        CPY #$06       schon 6 Stellen ?
CA0C  D0 DF        BNE $C9ED      nein
CA0E  20 E2 DA     JSR $DAE2      FAC = FAC * 10
CA11  20 9B DC     JSR $DC9B      FAC rechtsbündig machen
CA14  A6 64        LDX $64
```

```
CA16   A4 63       LDY $63         eingegebene Uhrzeit
CA18   A5 65       LDA $65
CA1A   4C DB FF    JMP $FFDB       TIME setzen

*****************************  Zeichen auf Ziffer prüfen
CA1D   B1 22       LDA ($22),Y     Zeichen holen
CA1F   20 80 00    JSR $0080       auf Ziffer prüfen
CA22   90 03       BCC $CA27       ja
CA24   4C 48 D2    JMP $D248       gibt 'illegal quantity'
CA27   E9 2F       SBC #$2F        von ASCII nach Hex umwandeln
CA29   4C 7E DD    JMP $DD7E       in FAC und ARG übertragen

*****************************  Wertzuweisung an normalen String
CA2C   A0 02       LDY #$02
CA2E   B1 64       LDA ($64),Y     Stringadresse high
CA30   C5 34       CMP $34         mit Stringanfangsadresse vergleichen
CA32   90 17       BCC $CA4B       kleiner, String steht innerhalb Programm
CA34   D0 07       BNE $CA3D
CA36   88          DEY
CA37   B1 64       LDA ($64),Y     Stringadresse low
CA39   C5 33       CMP $33         vergleichen
CA3B   90 0E       BCC $CA4B
CA3D   A4 65       LDY $65
CA3F   C4 2E       CPY $2E
CA41   90 08       BCC $CA4B
CA43   D0 0D       BNE $CA52
CA45   A5 64       LDA $64
CA47   C5 2D       CMP $2D
CA49   B0 07       BCS $CA52
CA4B   A5 64       LDA $64
CA4D   A4 65       LDY $65
CA4F   4C 68 CA    JMP $CA68
CA52   A0 00       LDY #$00
CA54   B1 64       LDA ($64),Y     Länge des Strings
CA56   20 75 D4    JSR $D475       prüft Speicherplatz, setzt Stringzeiger
CA59   A5 50       LDA $50
CA5B   A4 51       LDY $51
CA5D   85 6F       STA $6F
CA5F   84 70       STY $70
CA61   20 7A D6    JSR $D67A       String in Stringbereich übertragen
CA64   A9 61       LDA #$61
CA66   A0 00       LDY #$00
CA68   85 50       STA $50
CA6A   84 51       STY $51
CA6C   20 DB D6    JSR $D6DB       Descriptor aus Stringstack löschen
CA6F   A0 00       LDY #$00
CA71   B1 50       LDA ($50),Y     Länge
CA73   91 49       STA ($49),Y
CA75   C8          INY
CA76   B1 50       LDA ($50),Y     Adresse low
CA78   91 49       STA ($49),Y
CA7A   C8          INY
CA7B   B1 50       LDA ($50),Y     und Adresse high
CA7D   91 49       STA ($49),Y     in Variable bringen
CA7F   60          RTS

*****************************  BASIC-Befehl  PRINT#
CA80   20 86 CA    JSR $CA86       CMD-Befehl
CA83   4C B5 CB    JMP $CBB5       und CLRCH
```

```
*****************************   CMD-Befehl
CA86  20 9E D7   JSR $D79E      holt Byte-Wert
CA89  F0 05      BEQ $CA90      kein weiterer Kode ?
CA8B  A9 2C      LDA #$2C       ','
CA8D  20 FF CE   JSR $CEFF      prüft auf Kode
CA90  08         PHP
CA91  86 13      STX $13        Nummer des Ausgabegeräts setzen
CA93  20 15 E1   JSR $E115      CHKOUT setzt Ausgabegerät
CA96  28         PLP
CA97  4C A0 CA   JMP $CAA0      zum PRINT-Befehl

CA9A  20 21 CB   JSR $CB21      String drucken
CA9D  20 79 00   JSR $0079      CHRGOT  laufendes Zeichen holen

*****************************   BASIC-Befehl  PRINT
CAA0  F0 35      BEQ $CAD7
CAA2  F0 43      BEQ $CAE7
CAA4  C9 A3      CMP #$A3       'TAB(' - Kode
CAA6  F0 50      BEQ $CAF8
CAA8  C9 A6      CMP #$A6       'SPC(' - Kode
CAAA  18         CLC
CAAB  F0 4B      BEQ $CAF8
CAAD  C9 2C      CMP #$2C       ','
CAAF  F0 37      BEQ $CAE8
CAB1  C9 3B      CMP #$3B       ';'
CAB3  F0 5E      BEQ $CB13
CAB5  20 9E CD   JSR $CD9E      FRMEVL  Ausdruck holen
CAB8  24 0D      BIT $0D        Typflag
CABA  30 DE      BMI $CA9A      String ?
CABC  20 DD DD   JSR $DDDD      FAC in ASCII-String umwandeln
CABF  20 87 D4   JSR $D487      String-Parameter holen
CAC2  20 21 CB   JSR $CB21      String drucken
CAC5  20 3B CB   JSR $CB3B      Cursor RIGHT bzw. Leerzeichen ausgeben
CAC8  D0 D3      BNE $CA9D      weiter machen

CACA  A9 00      LDA #$00
CACC  9D 00 02   STA $0200,X    Eingabepuffer mit 0 abschließen
CACF  A2 FF      LDX #$FF
CAD1  A0 01      LDY #$01       Zeiger X/Y auf Eingabepuffer setzen
CAD3  A5 13      LDA $13        Ausgabe in File ?
CAD5  D0 10      BNE $CAE7
CAD7  A9 0D      LDA #$0D       'CR'
CAD9  20 47 CB   JSR $CB47      ausgeben
CADC  24 13      BIT $13        logische Filenummer
CADE  10 05      BPL $CAE5      kleiner 128 ?
CAE0  A9 0A      LDA #$0A       'LF' Zeilenvorschub
CAE2  20 47 CB   JSR $CB47      ausgeben
CAE5  49 FF      EOR #$FF
CAE7  60         RTS

CAE8  38         SEC            Zehner-Tabulator mit Komma
CAE9  20 F0 FF   JSR $FFF0      Cursorposition holen
CAEC  98         TYA
CAED  38         SEC
CAEE  E9 0B      SBC #$0B       10 abziehen
CAF0  B0 FC      BCS $CAEE      nicht negativ ?
CAF2  49 FF      EOR #$FF       invertieren
CAF4  69 01      ADC #$01
CAF6  D0 16      BNE $CB0E
```

```
******************************  TAB( (C=1) und SPC( (C=0)
CAF8  08          PHP          Kode merken
CAF9  38          SEC
CAFA  20 F0 FF    JSR $FFF0    Cursorposition holen
CAFD  84 09       STY $09
CAFF  20 9B D7    JSR $D79B    Byte-Wert holen
CB02  C9 29       CMP #$29     ')'  Klammer zu ?
CB04  D0 59       BNE $CB5F    nein, 'syntax error'
CB06  28          PLP
CB07  90 06       BCC $CB0F    zu SPC(
CB09  8A          TXA          TAB-Wert in Akku
CB0A  E5 09       SBC $09      mit Cursorposition vergleichen
CB0C  90 05       BCC $CB13    Wert kleiner, dann fertig
CB0E  AA          TAX
CB0F  E8          INX
CB10  CA          DEX
CB11  D0 06       BNE $CB19
CB13  20 73 00    JSR $0073    CHRGET  nächstes Zeichen holen
CB16  4C A2 CA    JMP $CAA2    und weiter machen
CB19  20 3B CB    JSR $CB3B    Cursor RIGHT bzw. Leerzeichen ausgeben
CB1C  D0 F2       BNE $CB10    zum Schleifenanfang
******************************  String ausgeben
CB1E  20 87 D4    JSR $D487    String-Parameter holen
CB21  20 A6 D6    JSR $D6A6    FRESTR
CB24  AA          TAX          Stringlänge
CB25  A0 00       LDY #$00
CB27  E8          INX
CB28  CA          DEX
CB29  F0 BC       BEQ $CAE7    String zu Ende
CB2B  B1 22       LDA ($22),Y  Zeichen des Strings
CB2D  20 47 CB    JSR $CB47    ausgeben
CB30  C8          INY
CB31  C9 0D       CMP #$0D     'CR' carriage return
CB33  D0 F3       BNE $CB28    nein, dann weiter
CB35  20 E5 CA    JSR $CAE5    Fehler ! Test auf LF  JSR $CADC
CB38  4C 28 CB    JMP $CB28    und weiter machen

******************************  Ausgabe eines Leerzeichens bzw. Cursor right
CB3B  A5 13       LDA $13      logische Filenummer
CB3D  F0 03       BEQ $CB42    keine Ausgabe in File
CB3F  A9 20       LDA #$20     dann Leerzeichen
CB41  2C          .BYTE $2C
CB42  A9 1D       LDA #$1D     Cursor right
CB44  2C          .BYTE $2C
CB45  A9 3F       LDA #$3F     '?' Fragezeichen
CB47  20 09 E1    JSR $E109    ausgeben
CB4A  29 FF       AND #$FF     Flags setzen
CB4C  60          RTS

******************************  Fehlerbehandlung bei Eingabe
CB4D  A5 11       LDA $11      Flag für INPUT / GET / READ
CB4F  F0 11       BEQ $CB62    INPUT
CB51  30 04       BMI $CB57    READ
CB53  A0 FF       LDY #$FF
CB55  D0 04       BNE $CB5B    GET

******************************  Fehler bei READ
CB57  A5 3F       LDA $3F
CB59  A4 40       LDY $40      DATA-Zeilennummer
```

```
*****************************  Fehler bei GET
CB5B  85 39        STA $39
CB5D  84 3A        STY $3A      gleich Zeilennummer des Fehlers
CB5F  4C 08 CF     JMP $CF08    gibt 'syntax error'

*****************************  Fehler bei INPUT
CB62  A5 13        LDA $13      Nummer des Eingabegeräts
CB64  F0 05        BEQ $CB6B    Eingabe vom Tastatur ?
CB66  A2 18        LDX #$18     Nummer für 'file data'
CB68  4C 37 C4     JMP $C437    Fehlermeldung ausgeben
CB6B  A9 0C        LDA #$0C
CB6D  A0 CD        LDY #$CD     Zeiger auf '?redo from start'
CB6F  20 1E CB     JSR $CB1E    String ausgeben
CB72  A5 3D        LDA $3D
CB74  A4 3E        LDY $3E      Programzeiger
CB76  85 7A        STA $7A      zurück auf INPUT-Befehl
CB78  84 7B        STY $7B
CB7A  60           RTS

*****************************  BASIC-Befehl  GET
CB7B  20 A6 D3     JSR $D3A6    Direktmodus ?
CB7E  C9 23        CMP #$23     '#'
CB80  D0 10        BNE $CB92    nein ?
CB82  20 73 00     JSR $0073    CHRGET  nächstes Zeichen holen
CB85  20 9E D7     JSR $D79E    Byte-Wert holen
CB88  A9 2C        LDA #$2C     ','
CB8A  20 FF CE     JSR $CEFF    prüft auf Kode
CB8D  86 13        STX $13      setzt Nummer des Eingabegeräts
CB8F  20 1B E1     JSR $E11B    CHKIN  setzt Eingabegerät
CB92  A2 01        LDX #$01
CB94  A0 02        LDY #$02     Zeiger X/Y auf $201, ein Zeichen
CB96  A9 00        LDA #$00     Puffer mit 0 abschließen
CB98  8D 01 02     STA $0201
CB9B  A9 40        LDA #$40     GET-Flag
CB9D  20 0F CC     JSR $CC0F    Eingabe und Wertzuweisung
CBA0  A6 13        LDX $13      Eingabe von File ?
CBA2  D0 13        BNE $CBB7    ja
CBA4  60           RTS          nein, dann fertig

*****************************  BASIC-Befehl  INPUT#
CBA5  20 9E D7     JSR $D79E    holt Byte-Wert
CBA8  A9 2C        LDA #$2C     ','
CBAA  20 FF CE     JSR $CEFF    prüft auf Kode
CBAD  86 13        STX $13      logische Filenummer des Eingabegeräts
CBAF  20 1B E1     JSR $E11B    CHKIN  setzt Eingabegerät
CBB2  20 CE CB     JSR $CBCE    INPUT ohne Dialogstring
CBB5  A5 13        LDA $13
CBB7  20 CC FF     JSR $FFCC    CLRCH  I/O-Kanäle rücksetzen
CBBA  A2 00        LDX #$00
CBBC  86 13        STX $13      Eingabegerät wieder Tastatur
CBBE  60           RTS

*****************************  BASIC-Befehl  INPUT
CBBF  C9 22        CMP #$22     '"' Hochkomma
CBC1  D0 0B        BNE $CBCE    nein
CBC3  20 BD CE     JSR $CEBD    Dialogstring holen
CBC6  A9 3B        LDA #$3B     ';'
CBC8  20 FF CE     JSR $CEFF    prüft auf Kode
CBCB  20 21 CB     JSR $CB21    String ausgeben
CBCE  20 A6 D3     JSR $D3A6    Direktmodus ?
```

```
CBD1  A9 2C     LDA #$2C     ',' Komma
CBD3  8D FF 01  STA $01FF    an Pufferstart schreiben
CBD6  20 F9 CB  JSR $CBF9    Fragezeichen ausgeben
CBD9  A5 13     LDA $13      Eingabe über logisches File
CBDB  F0 0D     BEQ $CBEA    nein
CBDD  20 B7 FF  JSR $FFB7    Status holen
CBE0  29 02     AND #$02     Time out
CBE2  F0 06     BEQ $CBEA    nein
CBE4  20 B5 CB  JSR $CBB5    ja, CLRCH, Tastatur wieder aktivieren
CBE7  4C F8 C8  JMP $C8F8    nächstes Statement ausführen
CBEA  AD 00 02  LDA $0200    erstes Zeichen aus Puffer holen
CBED  D0 1E     BNE $CC0D    nicht Ende ?
CBEF  A5 13     LDA $13      Eingabe
CBF1  D0 E3     BNE $CBD6    von logischem File ?
CBF3  20 06 C9  JSR $C906    nächstes Statement suchen
CBF6  4C FB C8  JMP $C8FB    Programmzeiger auf nächstes Statement

CBF9  A5 13     LDA $13      Eingabegerät
CBFB  D0 06     BNE $CC03    von logischem File
CBFD  20 45 CB  JSR $CB45    ? ausgeben
CC00  20 3B CB  JSR $CB3B    Cursor RIGHT ausgeben
CC03  4C 60 C5  JMP $C560    Eingabezeile holen

*****************************  BASIC-Befehl  READ
CC06  A6 41     LDX $41
CC08  A4 42     LDY $42      DATA-Zeiger holen
CC0A  A9 98     LDA #$98     READ-Flag
CC0C  2C        .BYTE $2C
CC0D  A9 00     LDA #$00     INPUT-Flag
CC0F  85 11     STA $11      merken
CC11  86 43     STX $43
CC13  84 44     STY $44      INPUT-Zeiger setzen
CC15  20 8B D0  JSR $D08B    Variable suchen
CC18  85 49     STA $49
CC1A  84 4A     STY $4A      Variablenadresse speichern
CC1C  A5 7A     LDA $7A
CC1E  A4 7B     LDY $7B      Programmzeiger
CC20  85 4B     STA $4B
CC22  84 4C     STY $4C      zwischenspeichern
CC24  A6 43     LDX $43
CC26  A4 44     LDY $44      INPUT-Zeiger
CC28  86 7A     STX $7A      gleich Programmzeiger
CC2A  84 7B     STY $7B
CC2C  20 79 00  JSR $0079    CHRGOT  laufendes Zeichen holen
CC2F  D0 20     BNE $CC51
CC31  24 11     BIT $11      Eingabeflag
CC33  50 0C     BVC $CC41    nicht GET ?
CC35  20 21 E1  JSR $E121    ein Zeichen holen
CC38  8D 00 02  STA $0200    und in Puffer schreiben
CC3B  A2 FF     LDX #$FF
CC3D  A0 01     LDY #$01
CC3F  D0 0C     BNE $CC4D
CC41  30 75     BMI $CCB8
CC43  A5 13     LDA $13      Eingabegerät
CC45  D0 03     BNE $CC4A    nicht von Tastatur ?
CC47  20 45 CB  JSR $CB45    '?' ausgeben
CC4A  20 F9 CB  JSR $CBF9    zweites Fragezeichen ausgeben
CC4D  86 7A     STX $7A
CC4F  84 7B     STY $7B      Programmzeiger setzen
CC51  20 73 00  JSR $0073    CHRGET  nächstes Zeichen holen
```

```
CC54  24 0D      BIT $0D        Typflag
CC56  10 31      BPL $CC89
CC58  24 11      BIT $11        Eingabeflag
CC5A  50 09      BVC $CC65
CC5C  E8         INX
CC5D  86 7A      STX $7A
CC5F  A9 00      LDA #$00
CC61  85 07      STA $07
CC63  F0 0C      BEQ $CC71
CC65  85 07      STA $07
CC67  C9 22      CMP #$22       '"' Hochkomma
CC69  F0 07      BEQ $CC72
CC6B  A9 3A      LDA #$3A       ':' Doppelpunkt
CC6D  85 07      STA $07
CC6F  A9 2C      LDA #$2C       ',' Komma
CC71  18         CLC
CC72  85 08      STA $08
CC74  A5 7A      LDA $7A
CC76  A4 7B      LDY $7B
CC78  69 00      ADC #$00
CC7A  90 01      BCC $CC7D
CC7C  C8         INY
CC7D  20 8D D4   JSR $D48D      String übernehmen
CC80  20 E2 D7   JSR $D7E2      Programmzeiger hinter String setzen
CC83  20 DA C9   JSR $C9DA      String an Variable zuweisen
CC86  4C 91 CC   JMP $CC91      weiter machen

CC89  20 F3 DC   JSR $DCF3      Ziffernstring in FAC holen
CC8C  A5 0E      LDA $0E        INTEGER-Flag
CC8E  20 C2 C9   JSR $C9C2      FAC an numerische Variable zuweisen
CC91  20 79 00   JSR $0079      CHRGOT  laufendes Zeichen holen
CC94  F0 07      BEQ $CC9D      Ende ?
CC96  C9 2C      CMP #$2C       ','
CC98  F0 03      BEQ $CC9D
CC9A  4C 4D CB   JMP $CB4D      zur Fehlerbehandlung

CC9D  A5 7A      LDA $7A
CC9F  A4 7B      LDY $7B        Programmzeiger
CCA1  85 43      STA $43
CCA3  84 44      STY $44        gleich DATA-Zeiger
CCA5  A5 4B      LDA $4B
CCA7  A4 4C      LDY $4C        Programmzeiger
CCA9  85 7A      STA $7A        wieder zurückholen
CCAB  84 7B      STY $7B
CCAD  20 79 00   JSR $0079      CHRGOT  laufendes Zeichen holen
CCB0  F0 2D      BEQ $CCDF
CCB2  20 FD CE   JSR $CEFD      CHKCOM  prüft auf Kamma
CCB5  4C 15 CC   JMP $CC15      weiter machen

CCB8  20 06 C9   JSR $C906      nächstes Statement suchen
CCBB  C8         INY
CCBC  AA         TAX
CCBD  D0 12      BNE $CCD1
CCBF  A2 0D      LDX #$0D       Nummer für 'out of data'
CCC1  C8         INY
CCC2  B1 7A      LDA ($7A),Y    Programmende ?
CCC4  F0 6C      BEQ $CD32      ja, dann Fehlermeldung ausgeben
CCC6  C8         INY
CCC7  B1 7A      LDA ($7A),Y
CCC9  85 3F      STA $3F
```

```
CCCB  C8          INY           Programmzeiger
CCCC  B1 7A       LDA ($7A),Y
CCCE  C8          INY           nach DATA-Zeiger
CCCF  85 40       STA $40
CCD1  20 FB C8    JSR $C8FB     Programmzeiger auf nächtes Statement
CCD4  20 79 00    JSR $0079     CHRGOT  laufendes Zeichen holen
CCD7  AA          TAX
CCD8  E0 83       CPX #$83      'DATA' - Kode
CCDA  D0 DC       BNE $CCB8     nein, weiter suchen
CCDC  4C 51 CC    JMP $CC51     Daten lesen
CCDF  A5 43       LDA $43
CCE1  A4 44       LDY $44       Input-Zeiger
CCE3  A6 11       LDX $11       Eingabe-Flag
CCE5  10 03       BPL $CCEA     kein DATA
CCE7  4C 27 C8    JMP $C827     DATA-Zeiger setzen
CCEA  A0 00       LDY #$00
CCEC  B1 43       LDA ($43),Y
CCEE  F0 0B       BEQ $CCFB
CCF0  A5 13       LDA $13
CCF2  D0 07       BNE $CCFB
CCF4  A9 FC       LDA #$FC
CCF6  A0 CC       LDY #$CC      Zeiger auf '?extra ignored'
CCF8  4C 1E CB    JMP $CB1E     String ausgeben
CCFB  60          RTS

******************************
CCFC  3F 45 58 54 52 41 20 49   '?extra ignored'
CD04  47 4E 4F 52 45 44 0D 00
CD0C  3F 52 45 44 4F 20 46 52   '?redo from start'
CD14  4F 4D 20 53 54 41 52 54
CD1C  0D 00

******************************  BASIC-Befehl  NEXT
CD1E  D0 04       BNE $AD24     folgt Variablenname ?
CD20  A0 00       LDY #$00
CD22  F0 03       BEQ $CD27
CD24  20 8B D0    JSR $D08B     sucht Variable
CD27  85 49       STA $49
CD29  84 4A       STY $4A       Variablenadresse
CD2B  20 8A C3    JSR $C38A     sucht FOR-NEXT-Schleife im Stack
CD2E  F0 05       BEQ $CD35     gefunden
CD30  A2 0A       LDX #$0A      Nummer für 'next without for'
CD32  4C 37 C4    JMP $C437     Fehlermeldung ausgeben
CD35  9A          TXS
CD36  8A          TXA
CD37  18          CLC
CD38  69 04       ADC #$04
CD3A  48          PHA
CD3B  69 06       ADC #$06
CD3D  85 24       STA $24
CD3F  68          PLA
CD40  A0 01       LDY #$01
CD42  20 A2 DB    JSR $DBA2     Variable vom Stack holen
CD45  BA          TSX
CD46  BD 09 01    LDA $0109,X
CD49  85 66       STA $66
CD4B  A5 49       LDA $49       Variablenadresse
CD4D  A4 4A       LDY $4A
CD4F  20 67 D8    JSR $D867     addiert STEP-Wert zu FAC
CD52  20 D0 DB    JSR $DBD0     FAC nach Variable bringen
```

```
CD55  A0 01     LDY #$01
CD57  20 5D DC  JSR $DC5D    FAC mit Schleifenendwert vergleichen
CD5A  BA        TSX
CD5B  38        SEC
CD5C  FD 09 01  SBC $0109,X
CD5F  F0 17     BEQ $CD78
CD61  BD 0F 01  LDA $010F,X
CD64  85 39     STA $39
CD66  BD 10 01  LDA $0110,X  Zeilennummer holen
CD69  85 3A     STA $3A
CD6B  BD 12 01  LDA $0112,X
CD6E  85 7A     STA $7A
CD70  BD 11 01  LDA $0111,X  Programmzeiger holen
CD73  85 7B     STA $7B
CD75  4C AE C7  JMP $C7AE    zur Interpreterschleife
CD78  8A        TXA
CD79  69 11     ADC #$11
CD7B  AA        TAX
CD7C  9A        TXS
CD7D  20 79 00  JSR $0079    CHRGOT  laufendes Zeichen holen
CD80  C9 2C     CMP #$2C
CD82  D0 F1     BNE $CD75
CD84  20 73 00  JSR $0073    CHRGET  nächstes Zeichen holen
CD87  20 24 CD  JSR $CD24    nächste NEXT-Variable

*****************************  FRMNUM  numerischen Ausdruck holen
CD8A  20 9E CD  JSR $CD9E    FRMEVL  Ausdruck holen

*****************************  prüft auf numerisch
CD8D  18        CLC
CD8E  24        .BYTE $24
*****************************  prüft auf String
CD8F  38        SEC
CD90  24 0D     BIT $0D      Typflag testen
CD92  30 03     BMI $CD97
CD94  B0 03     BCS $CD99
CD96  60        RTS
CD97  B0 FD     BCS $CD96
CD99  A2 16     LDX #$16     Nummer für 'type mismatch'
CD9B  4C 37 C4  JMP $C437    Fehlermeldung ausgeben

*****************************  FRMEVL  auswerten eines beliebigen Ausdrucks
CD9E  A6 7A     LDX $7A
CDA0  D0 02     BNE $CDA4
CDA2  C6 7B     DEC $7B      Programmzeiger um eins erniedrigen
CDA4  C6 7A     DEC $7A
CDA6  A2 00     LDX #$00
CDA8  24        .BYTE $24
CDA9  48        PHA
CDAA  8A        TXA
CDAB  48        PHA
CDAC  A9 01     LDA #$01
CDAE  20 FB C3  JSR $C3FB    prüft auf genügend Platz im Stack
CDB1  20 83 CE  JSR $CE83    nächstes Element holen
CDB4  A9 00     LDA #$00
CDB6  85 4D     STA $4D      Maske für Vergleichsoperator
CDB8  20 79 00  JSR $0079    CHRGOT  laufendes Zeichen holen
CDBB  38        SEC
CDBC  E9 B1     SBC #$B1
CDBE  90 17     BCC $CDD7
```

```
CDC0  C9 03     CMP #$03
CDC2  B0 13     BCS $CDD7
CDC4  C9 01     CMP #$01
CDC6  2A        ROL           Maske für kleiner, gleich, größer
CDC7  49 01     EOR #$01
CDC9  45 4D     EOR $4D
CDCB  C5 4D     CMP $4D
CDCD  90 61     BCC $CE30
CDCF  85 4D     STA $4D
CDD1  20 73 00  JSR $0073     CHRGET  nächstes Zeichen holen
CDD4  4C BB CD  JMP $CDBB     zurück
CDD7  A6 4D     LDX $4D
CDD9  D0 2C     BNE $CE07
CDDB  B0 7B     BCS $CE58
CDDD  69 07     ADC #$07
CDDF  90 77     BCC $CE58
CDE1  65 0D     ADC $0D
CDE3  D0 03     BNE $CDE8
CDE5  4C 3D D6  JMP $D63D     Stringverkettung
CDE8  69 FF     ADC #$FF
CDEA  85 22     STA $22
CDEC  0A        ASL
CDED  65 22     ADC $22       Kode mal 3
CDEF  A8        TAY
CDF0  68        PLA
CDF1  D9 80 C0  CMP $C080,Y   mit Hierachieflag vergleichen
CDF4  B0 67     BCS $CE5D
CDF6  20 8D CD  JSR $CD8D     prüft auf numerisch
CDF9  48        PHA
CDFA  20 20 CE  JSR $CE20     Operatoradresse und Operanden auf Stack
CDFD  68        PLA
CDFE  A4 4B     LDY $4B
CE00  10 17     BPL $CE19
CE02  AA        TAX
CE03  F0 56     BEQ $CE5B
CE05  D0 5F     BNE $CE66
CE07  46 0D     LSR $0D       Stringflag löschen
CE09  8A        TXA
CE0A  2A        ROL
CE0B  A6 7A     LDX $7A
CE0D  D0 02     BNE $CE11
CE0F  C6 7B     DEC $7B       Programmzeiger eins zurück
CE11  C6 7A     DEC $7A
CE13  A0 1B     LDY #$1B      Offset des Hierarchieflags
CE15  85 4D     STA $4D       Flag setzen
CE17  D0 D7     BNE $CDF0
CE19  D9 80 C0  CMP $C080,Y   mit Hierarchieflag vergleichen
CE1C  B0 48     BCS $CE66     größer, dann ARG vom Stack holen
CE1E  90 D9     BCC $CDF9     sonst ARG auf Stack und weitermachen
CE20  B9 82 C0  LDA $C082,Y
CE23  48        PHA           Operationsadresse auf Stack
CE24  B9 81 C0  LDA $C081,Y
CE27  48        PHA
CE28  20 33 CE  JSR $CE33     Operanden auf Stack
CE2B  A5 4D     LDA $4D
CE2D  4C A9 CD  JMP $CDA9     zum Schleifenanfang
CE30  4C 08 CF  JMP $CF08     gibt 'syntax error'
CE33  A5 66     LDA $66       Vorzeichen
CE35  BE 80 C0  LDX $C080,Y   Hierarchieflag
CE38  A8        TAY
```

```
CE39  68        PLA
CE3A  85 22     STA $22        Rücksprungadresse merken
CE3C  E6 22     INC $22
CE3E  68        PLA
CE3F  85 23     STA $23
CE41  98        TYA
CE42  48        PHA
CE43  20 1B DC  JSR $DC1B      FAC runden
CE46  A5 65     LDA $65
CE48  48        PHA
CE49  A5 64     LDA $64
CE4B  48        PHA
CE4C  A5 63     LDA $63        FAC auf Stack
CE4E  48        PHA
CE4F  A5 62     LDA $62
CE51  48        PHA
CE52  A5 61     LDA $61
CE54  48        PHA
CE55  6C 22 00  JMP ($0022)    Sprung auf Operation
CE58  A0 FF     LDY #$FF
CE5A  68        PLA
CE5B  F0 23     BEQ $CE80
CE5D  C9 64     CMP #$64
CE5F  F0 03     BEQ $CE64
CE61  20 8D CD  JSR $CD8D      prüft auf numerisch
CE64  84 4B     STY $4B
CE66  68        PLA
CE67  4A        LSR
CE68  85 12     STA $12
CE6A  68        PLA
CE6B  85 69     STA $69
CE6D  68        PLA
CE6E  85 6A     STA $6A        ARG vom Stack holen
CE70  68        PLA
CE71  85 6B     STA $6B
CE73  68        PLA
CE74  85 6C     STA $6C
CE76  68        PLA
CE77  85 6D     STA $6D
CE79  68        PLA
CE7A  85 6E     STA $6E
CE7C  45 66     EOR $66
CE7E  85 6F     STA $6F
CE80  A5 61     LDA $61
CE82  60        RTS

******************************  Nächstes Element eines Ausdrucks holen
CE83  6C 0A 03  JMP ($030A)    JMP $CE86
CE86  A9 00     LDA #$00
CE88  85 0D     STA $0D        Typflag auf numerisch
CE8A  20 73 00  JSR $0073      CHRGET  nächstes Zeichen holen
CE8D  B0 03     BCS $CE92      Ziffer ?
CE8F  4C F3 DC  JMP $DCF3      Variable in FAC holen
CE92  20 13 D1  JSR $D113      Buchstabe ?
CE95  90 03     BCC $CE9A      nein
CE97  4C 28 CF  JMP $CF28      Variable holen
CE9A  C9 FF     CMP #$FF       BASIC-Kode für Pi ?
CE9C  D0 0F     BNE $CEAD
CE9E  A9 A8     LDA #$A8       Zeiger auf Konstante Pi
CEA0  A0 CE     LDY #$CE
```

```
CEA2  20 A2 DB    JSR $DBA2     Konstante in FAC holen
CEA5  4C 73 00    JMP $0073     nächstes Zeichen holen

******************************
CEA8  82 49 0F DA A1            Konstante Pi  3.14159265

CEAD  C9 2E       CMP #$2E      '.'
CEAF  F0 DE       BEQ $CE8F
CEB1  C9 AB       CMP #$AB      '-'  Interpreter-Kode
CEB3  F0 58       BEQ $CF0D
CEB5  C9 AA       CMP #$AA      '+'  Interpreter-Kode
CEB7  F0 D1       BEQ $CE8A
CEB9  C9 22       CMP #$22      '"'  Hochkomma
CEBB  D0 0F       BNE $CECC
CEBD  A5 7A       LDA $7A
CEBF  A4 7B       LDY $7B       Programzeiger holen
CEC1  69 00       ADC #$00
CEC3  90 01       BCC $CEC6
CEC5  C8          INY
CEC6  20 87 D4    JSR $D487     String übertragen
CEC9  4C E2 D7    JMP $D7E2     Programmzeiger auf Stringende + 1
CECC  C9 A8       CMP #$A8      'NOT' - Kode
CECE  D0 13       BNE $CEE3
CED0  A0 18       LDY #$18      Offset des Hierarchiekodes in Tabelle
CED2  D0 3B       BNE $CF0F

******************************  BASIC-Operator  NOT
CED4  20 BF D1    JSR $D1BF     FAC nach Integer wandlen
CED7  A5 65       LDA $65
CED9  49 FF       EOR #$FF      alle Bits umdrehen
CEDB  A8          TAY
CEDC  A5 64       LDA $64
CEDE  49 FF       EOR #$FF
CEE0  4C 91 D3    JMP $D391     wieder nach Fließkomma wandeln

******************************
CEE3  C9 A5       CMP #$A5      'FN' - Kode
CEE5  D0 03       BNE $CEEA
CEE7  4C F4 D3    JMP $D3F4     FN ausführen

******************************
CEEA  C9 B4       CMP #$B4      'SGN' - Kode
CEEC  90 03       BCC $CEF1     kleiner (keine Funktion) ?
CEEE  4C A7 CF    JMP $CFA7     zur Funktionsberechnung

******************************  holt Ausdruck in Klammern
CEF1  20 FA CE    JSR $CEFA     prüft auf '(' Klammer auf
CEF4  20 9E CD    JSR $CD9E     FRMEVL  holt Ausdruck
CEF7  A9 29       LDA #$29      ')' Klammer zu
CEF9  2C          .BYTE $2C
CEFA  A9 28       LDA #$28      '(' Klammer auf
CEFC  2C          .BYTE $2C
CEFD  A9 2C       LDA #$2C      ',' Komma
CEFF  A0 00       LDY #$00
CF01  D1 7A       CMP ($7A),Y   mit laufendem Zeichen vergleichen
CF03  D0 03       BNE $CF08     keine übereinstimmung ?
CF05  4C 73 00    JMP $0073     CHRGET nächstes Zeichen holen
CF08  A2 0B       LDX #$0B      Nummer für 'syntax error'
CF0A  4C 37 C4    JMP $C437     Fehlermeldung ausgeben
```

```
CF0D  A0 15     LDY #$15     Offset Hierarchiekode für Vorzeichenwechsel
CF0F  68        PLA
CF10  68        PLA
CF11  4C FA CD  JMP $CDFA    zur Auswertung

*****************************  prüft auf Systemvariable
CF14  38        SEC
CF15  A5 64     LDA $64      Variablenzeiger innerhalb des
CF17  E9 00     SBC #$00     BASIC-Interpreters ?
CF19  A5 65     LDA $65
CF1B  E9 C0     SBC #$C0     (Adresse $C000 bis $E387)
CF1D  90 08     BCC $CF27
CF1F  A9 87     LDA #$87     ja, dann C=1, sonst C=0
CF21  E5 64     SBC $64
CF23  A9 E3     LDA #$E3
CF25  E5 65     SBC $65
CF27  60        RTS

*****************************  Variable holen
CF28  20 8B D0  JSR $D08B    Variable suchen
CF2B  85 64     STA $64
CF2D  84 65     STY $65      zeigt auf Variable bzw. Stringdescriptor
CF2F  A6 45     LDX $45
CF31  A4 46     LDY $46
CF33  A5 0D     LDA $0D      Typflag
CF35  F0 26     BEQ $CF5D    numerisch ?
CF37  A9 00     LDA #$00
CF39  85 70     STA $70
CF3B  20 14 CF  JSR $CF14    Descriptor im Interpreter ?
CF3E  90 1C     BCC $CF5C    nein, dann fertig
CF40  E0 54     CPX #$54     'T'
CF42  D0 18     BNE $CF5C
CF44  C0 C9     CPY #$C9     'I$'
CF46  D0 14     BNE $CF5C
CF48  20 84 CF  JSR $CF84    Zeit holen
CF4B  84 5E     STY $5E
CF4D  88        DEY
CF4E  84 71     STY $71
CF50  A0 06     LDY #$06     Länge TI$
CF52  84 5D     STY $5D
CF54  A0 24     LDY #$24
CF56  20 68 DE  JSR $DE68    TI$ erzeugen
CF59  4C 6F D4  JMP $D46F
CF5C  60        RTS

CF5D  24 0E     BIT $0E      Integer/Real-Flag
CF5F  10 0D     BPL $CF6E    Real ?
CF61  A0 00     LDY #$00
CF63  B1 64     LDA ($64),Y
CF65  AA        TAX
CF66  C8        INY          Integerzahl holen
CF67  B1 64     LDA ($64),Y
CF69  A8        TAY
CF6A  8A        TXA
CF6B  4C 91 D3  JMP $D391    und nach Fließkomma wandeln

CF6E  20 14 CF  JSR $CF14    steht Variable im Interpreter ?
CF71  90 2D     BCC $CFA0    ja, dann Variablenwert holen
CF73  E0 54     CPX #$54     'T'
CF75  D0 1B     BNE $CF92
```

```
CF77  C0 49      CPY #$49      'I'
CF79  D0 25      BNE $CFA0
CF7B  20 84 CF   JSR $CF84     Zeit in FAC holen
CF7E  98         TYA
CF7F  A2 A0      LDX #$A0      Adresse low Byte von TI
CF81  4C 4F DC   JMP $DC4F     nach FAC wandeln

*****************************  Zeit holen
CF84  20 DE FF   JSR $FFDE     TIME nach A/X/Y holen
CF87  86 64      STX $64
CF89  84 63      STY $63
CF8B  85 65      STA $65
CF8D  A0 00      LDY #$00      und in Fließkommaakku
CF8F  84 62      STY $62
CF91  60         RTS

*****************************
CF92  E0 53      CPX #$53      'S'
CF94  D0 0A      BNE $CFA0
CF96  C0 54      CPY #$54      'T'
CF98  D0 06      BNE $CFA0
CF9A  20 B7 FF   JSR $FFB7     Status holen
CF9D  4C 3C DC   JMP $DC3C     Byte-Wert nach Fließkomma wandeln

*****************************  REAL-Variable holen
CFA0  A5 64      LDA $64
CFA2  A4 65      LDY $65       Variablenadresse
CFA4  4C A2 DB   JMP $DBA2     Variable in FAC holen

CFA7  0A         ASL           Kode mal 2
CFA8  48         PHA
CFA9  AA         TAX
CFAA  20 73 00   JSR $0073     CHRGET  nächstes Zeichen holen
CFAD  E0 8F      CPX #$8F
CFAF  90 20      BCC $CFD1     numerische Funktion ?
CFB1  20 FA CE   JSR $CEFA     prüft auf '(' Klammer auf
CFB4  20 9E CD   JSR $CD9E     FRMEVL  holt Ausdruck
CFB7  20 FD CE   JSR $CEFD     CHKCOM  prüft auf Komma
CFBA  20 8F CD   JSR $CD8F     prüft auf String
CFBD  68         PLA
CFBE  AA         TAX
CFBF  A5 65      LDA $65
CFC1  48         PHA           Adresse des Stringdescriptors
CFC2  A5 64      LDA $64
CFC4  48         PHA
CFC5  8A         TXA
CFC6  48         PHA
CFC7  20 9E D7   JSR $D79E     holt Byte-Wert
CFCA  68         PLA
CFCB  A8         TAY
CFCC  8A         TXA
CFCD  48         PHA
CFCE  4C D6 CF   JMP $CFD6     Routine ausführen

CFD1  20 F1 CE   JSR $CEF1     holt Term in Klammern
CFD4  68         PLA
CFD5  A8         TAY
CFD6  B9 EA ^F   LDA $BFEA,Y
CFD9  85 55      STA $55       setzt Vektor für
CFDB  B9 EB BF   LDA $BFEB,Y   Funktionsberechnung
```

```
CFDE  85 56       STA $56
CFE0  20 54 00    JSR $0054     Funktion ausführen
CFE3  4C 8D CD    JMP $CD8D     prüft auf numerisch

****************************    BASIC-Operator  OR
CFE6  A0 FF       LDY #$FF      Flag für OR
CFE8  2C          .BYTE $2C

****************************    BASIC-Operator  AND
CFE9  A0 00       LDY #$00      Flag für AND
CFEB  84 0B       STY $0B       speichern
CFED  20 BF D1    JSR $D1BF     FAC nach Integer wandeln
CFF0  A5 64       LDA $64
CFF2  45 0B       EOR $0B
CFF4  85 07       STA $07
CFF6  A5 65       LDA $65       mit Flag verküpfen und nach $7/$8
CFF8  45 0B       EOR $0B
CFFA  85 08       STA $08
CFFC  20 FC DB    JSR $DBFC     ARG nach FAC
CFFF  20 BF D1    JSR $D1BF     FAC nach Integer wandeln
D002  A5 65       LDA $65
D004  45 0B       EOR $0B
D006  25 08       AND $08       logische Verknüpfung
D008  45 0B       EOR $0B
D00A  A8          TAY
D00B  A5 64       LDA $64
D00D  45 0B       EOR $0B
D00F  25 07       AND $07
D011  45 0B       EOR $0B
D013  4C 91 D3    JMP $D391     wieder nach Fließkomma wandeln

****************************    Vergleich
D016  20 90 CD    JSR $CD90     prüft auf identischen Variablentyp
D019  B0 13       BCS $D02E     String ? dann weiter
D01B  A5 6E       LDA $6E
D01D  09 7F       ORA #$7F      ARG in Speicherformat
D01F  25 6A       AND $6A
D021  85 6A       STA $6A
D023  A9 69       LDA #$69      Zeiger A/Y auf ARG
D025  A0 00       LDY #$00
D027  20 5B DC    JSR $DC5B     Vergleich ARG mit FAC
D02A  AA          TAX
D02B  4C 61 D0    JMP $D061     Ergebnis in FAC holen

****************************    Stringvergleich
D02E  A9 00       LDA #$00
D030  85 0D       STA $0D       Stringflag löschen
D032  C6 4D       DEC $4D
D034  20 A6 D6    JSR $D6A6     FRESTR
D037  85 61       STA $61       Stringlänge
D039  86 62       STX $62
D03B  84 63       STY $63       Stringadresse
D03D  A5 6C       LDA $6C
D03F  A4 6D       LDY $6D       Zeiger auf zweiten String
D041  20 AA D6    JSR $D6AA     FRESTR
D044  86 6C       STX $6C
D046  84 6D       STY $6D       Stringadresse des 2. Strings
D048  AA          TAX
D049  38          SEC
D04A  E5 61       SBC $61
```

```
D04C  F0 08       BEQ $D056      Längen vergleichen
D04E  A9 01       LDA #$01
D050  90 04       BCC $D056      2. String kürzer
D052  A6 61       LDX $61        Länge des ersten Strings
D054  A9 FF       LDA #$FF
D056  85 66       STA $66
D058  A0 FF       LDY #$FF
D05A  E8          INX
D05B  C8          INY
D05C  CA          DEX
D05D  D0 07       BNE $D066
D05F  A6 66       LDX $66
D061  30 0F       BMI $D072
D063  18          CLC
D064  90 0C       BCC $D072
D066  B1 6C       LDA ($6C),Y    zeichenweiser Vergleich
D068  D1 62       CMP ($62),Y    der Strings
D06A  F0 EF       BEQ $D05B
D06C  A2 FF       LDX #$FF
D06E  B0 02       BCS $D072
D070  A2 01       LDX #$01
D072  E8          INX
D073  8A          TXA
D074  2A          ROL
D075  25 12       AND $12
D077  F0 02       BEQ $D07B
D079  A9 FF       LDA #$FF
D07B  4C 3C DC    JMP $DC3C      Ergebnis nach FAC holen

D07E  20 FD CE    JSR $CEFD      CHKCOM  prüft auf Komma

*****************************    BASIC-Befehl  DIM
D081  AA          TAX
D082  20 90 D0    JSR $D090      Variable dimensionieren
D085  20 79 00    JSR $0079      CHRGOT  laufendes Zeichen holen
D088  D0 F4       BNE $D07E      nicht Ende, dann zur nächsten Variablen
D08A  60          RTS

D08B  A2 00       LDX #$00       Flag für nicht dimensioniert
D08D  20 79 00    JSR $0079      CHRGOT  laufendes Zeichen holen
D090  86 0C       STX $0C        DIM-Flag setzen
D092  85 45       STA $45        Variablenname
D094  20 79 00    JSR $0079      CHRGOT  laufendes Zeichen holen
D097  20 13 D1    JSR $D113      prüft auf Buchstabe
D09A  B0 03       BCS $D09F      ja
D09C  4C 08 CF    JMP $CF08      gibt 'syntax error'

D09F  A2 00       LDX #$00
D0A1  86 0D       STX $0D        Typflag löschen
D0A3  86 0E       STX $0E        Integerflag löschen
D0A5  20 73 00    JSR $0073      CHRGET  nächstes Zeichen holen
D0A8  90 05       BCC $D0AF      Ziffer ?
D0AA  20 13 D1    JSR $D113      prüft auf Buchstabe
D0AD  90 0B       BCC $D0BA      nein
D0AF  AA          TAX
D0B0  20 73 00    JSR $0073      CHRGET  nächstes Zeichen holen
D0B3  90 FB       BCC $D0B0      Ziffer ?
D0B5  20 13 D1    JSR $D113      prüft auf Buchstabe
D0B8  B0 F6       BCS $D0B0      ja
D0BA  C9 24       CMP #$24       '$'
```

```
D0BC  D0 06     BNE $D0C4    nein
D0BE  A9 FF     LDA #$FF
D0C0  85 0D     STA $0D      Stringflag setzen
D0C2  D0 10     BNE $D0D4
D0C4  C9 25     CMP #$25     '%'
D0C6  D0 13     BNE $D0DB    nein
D0C8  A5 10     LDA $10      Integer erlaubt ?
D0CA  D0 D0     BNE $D09C    nein, 'syntax error'
D0CC  A9 80     LDA #$80
D0CE  85 0E     STA $0E      Integerflag setzen
D0D0  05 45     ORA $45      Bit 7 im Namen setzen
D0D2  85 45     STA $45
D0D4  8A        TXA
D0D5  09 80     ORA #$80
D0D7  AA        TAX
D0D8  20 73 00  JSR $0073    CHRGET  nächstes Zeichen holen
D0DB  86 46     STX $46      zweiten Buchstaben des Namens speichern
D0DD  38        SEC
D0DE  05 10     ORA $10
D0E0  E9 28     SBC #$28     '('
D0E2  D0 03     BNE $D0E7    nicht Klammer auf ?
D0E4  4C D1 D1  JMP $D1D1    dimensionierte Variable verarbeiten

D0E7  A0 00     LDY #$00
D0E9  84 10     STY $10
D0EB  A5 2D     LDA $2D      Zeiger auf Variablenanfang
D0ED  A6 2E     LDX $2E
D0EF  86 60     STX $60
D0F1  85 5F     STA $5F      zum Suchen merken
D0F3  E4 30     CPX $30
D0F5  D0 04     BNE $D0FB
D0F7  C5 2F     CMP $2F      Ende der Varibalen schon erreicht ?
D0F9  F0 22     BEQ $D11D    ja, nicht gefunden
D0FB  A5 45     LDA $45      ersten Buchstaben des Namens
D0FD  D1 5F     CMP ($5F),Y  mit Variablentabelle vergleichen
D0FF  D0 08     BNE $D109    nein, weitersuchen
D101  A5 46     LDA $46      zweiten Buchstaben
D103  C8        INY
D104  D1 5F     CMP ($5F),Y  vergleichen
D106  F0 7D     BEQ $D185
D108  88        DEY
D109  18        CLC
D10A  A5 5F     LDA $5F
D10C  69 07     ADC #$07     Zeiger um sieben erhöhen (2 + 5 Byte)
D10E  90 E1     BCC $D0F1
D110  E8        INX
D111  D0 DC     BNE $D0EF    weiter suchen

*****************************  prüft auf Buchstabe
D113  C9 41     CMP #$41     'A'
D115  90 05     BCC $D11C
D117  E9 5B     SBC #$5B     'Z'
D119  38        SEC          ja, dann C=1
D11A  E9 A5     SBC #$A5     nein, dann C=0
D11C  60        RTS

*****************************
D11D  68        PLA
D11E  48        PHA          Aufrufadresse prüfen
D11F  C9 2A     CMP #$2A     Aufruf von FRMEVL ?
```

```
D121  D0 05     BNE $D128     nein
D123  A9 13     LDA #$13
D125  A0 DF     LDY #$DF      Zeiger auf Konstante 0
D127  60        RTS
D128  A5 45     LDA $45
D12A  A4 46     LDY $46       Varibalenname
D12C  C9 54     CMP #$54      'T'
D12E  D0 0B     BNE $D13B
D130  C0 C9     CPY #$C9      'I$'
D132  F0 EF     BEQ $D123     ja ,TI$
D134  C0 49     CPY #$49      'I'
D136  D0 03     BNE $D13B     nein
D138  4C 08 CF  JMP $CF08     gibt 'syntax error'
D13B  C9 53     CMP #$53      'S'
D13D  D0 04     BNE $D143
D13F  C0 54     CPY #$54      'T'
D141  F0 F5     BEQ $D138     ST, dann 'syntax error'
D143  A5 2F     LDA $2F
D145  A4 30     LDY $30       Zeiger auf Arraytabelle
D147  85 5F     STA $5F
D149  84 60     STY $60       merken
D14B  A5 31     LDA $31
D14D  A4 32     LDY $32       Zeiger auf Ende der Arraytabelle
D14F  85 5A     STA $5A
D151  84 5B     STY $5B       merken
D153  18        CLC
D154  69 07     ADC #$07      um 7 verschieben für Anlage einer
D156  90 01     BCC $D159     neuen Varaibalen
D158  C8        INY
D159  85 58     STA $58
D15B  84 59     STY $59       neues Blockende
D15D  20 B8 C3  JSR $C3B8     Block verschieben
D160  A5 58     LDA $58
D162  A4 59     LDY $59
D164  C8        INY
D165  85 2F     STA $2F       Zeiger auf Arraytabelle neu setzen
D167  84 30     STY $30
D169  A0 00     LDY #$00
D16B  A5 45     LDA $45       erster Buchstabe des Namens
D16D  91 5F     STA ($5F),Y
D16F  C8        INY
D170  A5 46     LDA $46       zweiter Buchstabe des Namens
D172  91 5F     STA ($5F),Y
D174  A9 00     LDA #$00
D176  C8        INY
D177  91 5F     STA ($5F),Y
D179  C8        INY
D17A  91 5F     STA ($5F),Y
D17C  C8        INY           5mal Null als Variablenwert
D17D  91 5F     STA ($5F),Y
D17F  C8        INY
D180  91 5F     STA ($5F),Y
D182  C8        INY
D183  91 5F     STA ($5F),Y
D185  A5 5F     LDA $5F
D187  18        CLC
D188  69 02     ADC #$02
D18A  A4 60     LDY $60
D18C  90 01     BCC $D18F
D18E  C8        INY
```

```
D18F  85 47        STA $47
D191  84 48        STY $48
D193  60           RTS

*****************************  berechnet Zeiger auf erstes Arrayelement
D194  A5 0B        LDA $0B     Anzahl der Dimensionen
D196  0A           ASL         mal 2
D197  69 05        ADC #$05    plus 5
D199  65 5F        ADC $5F
D19B  A4 60        LDY $60     zu $5F/$60 addieren
D19D  90 01        BCC $D1A0
D19F  C8           INY
D1A0  85 58        STA $58     Ergebnis-Zeiger
D1A2  84 59        STY $59
D1A4  60           RTS

*****************************
D1A5  90 80 00 00 00           Fließkommakonstante -32768

*****************************
D1AA  20 BF D1     JSR $D1BF   FAC nach Integer wamdeln
D1AD  A5 64        LDA $64
D1AF  A4 65        LDY $65
D1B1  60           RTS

*****************************  Umwandlung Fließkomma nach Integer
D1B2  20 73 00     JSR $0073   CHRGET  nächstes Zeichen holen
D1B5  20 9E CD     JSR $CD9E   FRMEVL  Ausdruck holen
D1B8  20 8D CD     JSR $CD8D   prüft auf numerisch
D1BB  A5 66        LDA $66     Vorzeichen
D1BD  30 0D        BMI $D1CC   negativ, dann 'illegal quantity'
D1BF  A5 61        LDA $61     Exponent
D1C1  C9 90        CMP #$90    Betrag größer 32768 ?
D1C3  90 09        BCC $D1CE   nein
D1C5  A9 A5        LDA #$A5
D1C7  A0 D1        LDY #$D1    Zeiger auf Konstante -32768
D1C9  20 5B DC     JSR $DC5B   mit FAC vergleichen
D1CC  D0 7A        BNE $D248   ungleich, dann 'illegal qunatity'
D1CE  4C 9B DC     JMP $DC9B   wandelt Fließkomma nach Integer

*****************************  dimensionierte Variable
D1D1  A5 0C        LDA $0C     DIM-Flag
D1D3  05 0E        ORA $0E     Integer-Flag
D1D5  48           PHA
D1D6  A5 0D        LDA $0D     Typ-Flag
D1D8  48           PHA
D1D9  A0 00        LDY #$00
D1DB  98           TYA
D1DC  48           PHA
D1DD  A5 46        LDA $46     zweiter Buchstabe des Namens
D1DF  48           PHA
D1E0  A5 45        LDA $45     erster Buchstabe des Variablennamens
D1E2  48           PHA
D1E3  20 B2 D1     JSR $D1B2   Index holen und nach Integer
D1E6  68           PLA
D1E7  85 45        STA $45
D1E9  68           PLA         Variablenname zurückholen
D1EA  85 46        STA $46
D1EC  68           PLA
D1ED  A8           TAY
```

```
D1EE  BA          TSX
D1EF  BD 02 01    LDA $0102,X
D1F2  48          PHA            Flags vom Stack holen
D1F3  BD 01 01    LDA $0101,X
D1F6  48          PHA
D1F7  A5 64       LDA $64
D1F9  9D 02 01    STA $0102,X    Index low und high auf Stack
D1FC  A5 65       LDA $65
D1FE  9D 01 01    STA $0101,X
D201  C8          INY
D202  20 79 00    JSR $0079      CHRGOT  laufendes Zeichen holen
D205  C9 2C       CMP #$2C       ','  Komma
D207  F0 D2       BEQ $D1DB      ja, dann nächsten Index
D209  84 0B       STY $0B        Anzahl der Indizes
D20B  20 F7 CE    JSR $CEF7      prüft auf Klammer zu
D20E  68          PLA
D20F  85 0D       STA $0D
D211  68          PLA            Flags zurückholen
D212  85 0E       STA $0E
D214  29 7F       AND #$7F
D216  85 0C       STA $0C
D218  A6 2F       LDX $2F
D21A  A5 30       LDA $30        Zeiger auf Arraytabelle
D21C  86 5F       STX $5F
D21E  85 60       STA $60        Zeiger merken
D220  C5 32       CMP $32
D222  D0 04       BNE $D228
D224  E4 31       CPX $31        mit Tabellenende vergleichen
D226  F0 39       BEQ $D261      ja, nicht gefunden
D228  A0 00       LDY #$00
D22A  B1 5F       LDA ($5F),Y    Namen aus Tabelle
D22C  C8          INY
D22D  C5 45       CMP $45        mit gesuchtem Namen vergleichen
D22F  D0 06       BNE $D237
D231  A5 46       LDA $46        zweiten Buchstaben
D233  D1 5F       CMP ($5F),Y    vergleichen
D235  F0 16       BEQ $D24D      gefunden
D237  C8          INY
D238  B1 5F       LDA ($5F),Y
D23A  18          CLC
D23B  65 5F       ADC $5F        Feldlänge addieren
D23D  AA          TAX
D23E  C8          INY
D23F  B1 5F       LDA ($5F),Y
D241  65 60       ADC $60
D243  90 D7       BCC $D21C      und weiter suchen

D245  A2 12       LDX #$12       Nummer für 'bad subscript'
D247  2C          .BYTE $2C
D248  A2 0E       LDX #$0E       Nummer für 'illegal quantity'
D24A  4C 37 C4    JMP $C437      Fehlermeldung ausgeben

D24D  A2 13       LDX #$13       Nummer für 'redim'd array'
D24F  A5 0C       LDA $0C        DIM-Flag null ?
D251  D0 F7       BNE $D24A      nein, dann Fehlermeldung
D253  20 94 D1    JSR $D194      setzt Zeiger auf erstes Arrayelement
D256  A5 0B       LDA $0B        Zahl der gefundenen Dimensionen
D258  A0 04       LDY #$04
D25A  D1 5F       CMP ($5F),Y    mit Dimensionen des Arrays vergleichen
D25C  D0 E7       BNE $D245      ungleich, dann 'bad subscript'
```

```
D25E  4C EA D2    JMP $D2EA      sucht gewünschtes Arrayelement

*****************************   Arrayvariable anlegen
D261  20 94 D1    JSR $D194      Länge des Arraykopfs berechnen
D264  20 08 C4    JSR $C408      prüft auf genügend Platz im Speicher
D267  A0 00       LDY #$00
D269  84 72       STY $72
D26B  A2 05       LDX #$05       Defaultwert für Variablenlänge (Real)
D26D  A5 45       LDA $45        erster Buchstabe des Namens
D26F  91 5F       STA ($5F),Y    in Arraytabelle
D271  10 01       BPL $D274      kein Integer ?
D273  CA          DEX
D274  C8          INY
D275  A5 46       LDA $46        zweiter Buchstabe
D277  91 5F       STA ($5F),Y    in Tabelle schreiben
D279  10 02       BPL $D27D      kein String oder Integer ?
D27B  CA          DEX
D27C  CA          DEX
D27D  86 71       STX $71        endgültige Variablenlänge (2, 3 oder 5)
D27F  A5 0B       LDA $0B        Anzahl der Dimensionen
D281  C8          INY
D282  C8          INY
D283  C8          INY
D284  91 5F       STA ($5F),Y    speichern
D286  A2 0B       LDX #$0B       11, Default für Dimensionierung
D288  A9 00       LDA #$00
D28A  24 0C       BIT $0C        Aufruf durch DIM-Befehl ?
D28C  50 08       BVC $D296      nein
D28E  68          PLA            Dimension vom Stack holen
D28F  18          CLC
D290  69 01       ADC #$01       eins addieren für Nullelement
D292  AA          TAX
D293  68          PLA
D294  69 00       ADC #$00
D296  C8          INY
D297  91 5F       STA ($5F),Y
D299  C8          INY
D29A  8A          TXA            und speichern
D29B  91 5F       STA ($5F),Y
D29D  20 4C D3    JSR $D34C      Platz für weitere Dimensionen berechnen
D2A0  86 71       STX $71
D2A2  85 72       STA $72        Variablenendezeiger merken
D2A4  A4 22       LDY $22
D2A6  C6 0B       DEC $0B        weitere Dimensionen
D2A8  D0 DC       BNE $D286      ja
D2AA  65 59       ADC $59
D2AC  B0 5D       BCS $D30B      Feldlänge plus Startadresse
D2AE  85 59       STA $59
D2B0  A8          TAY
D2B1  8A          TXA
D2B2  65 58       ADC $58
D2B4  90 03       BCC $D2B9
D2B6  C8          INY
D2B7  F0 52       BEQ $D30B
D2B9  20 08 C4    JSR $C408      prüft auf genügend Speicherplatz
D2BC  85 31       STA $31
D2BE  84 32       STY $32        Zeiger auf Ende der Arraytabelle
D2C0  A9 00       LDA #$00       Array mit Nullen füllen
D2C2  E6 72       INC $72
D2C4  A4 71       LDY $71
```

```
D2C6  F0 05       BEQ $D2CD
D2C8  88          DEY
D2C9  91 58       STA ($58),Y
D2CB  D0 FB       BNE $D2C8
D2CD  C6 59       DEC $59
D2CF  C6 72       DEC $72
D2D1  D0 F5       BNE $D2C8
D2D3  E6 59       INC $59
D2D5  38          SEC
D2D6  A5 31       LDA $31
D2D8  E5 5F       SBC $5F
D2DA  A0 02       LDY #$02
D2DC  91 5F       STA ($5F),Y   Arraylänge low
D2DE  A5 32       LDA $32
D2E0  C8          INY
D2E1  E5 60       SBC $60
D2E3  91 5F       STA ($5F),Y   Arraylänge high
D2E5  A5 0C       LDA $0C       Aufruf durch DIM-Befehl ?
D2E7  D0 62       BNE $D34B     ja, RTS

******************************  Arrayelement suchen
D2E9  C8          INY
D2EA  B1 5F       LDA ($5F),Y   Zahl der Dimensionen
D2EC  85 0B       STA $0B
D2EE  A9 00       LDA #$00
D2F0  85 71       STA $71
D2F2  85 72       STA $72
D2F4  C8          INY
D2F5  68          PLA
D2F6  AA          TAX
D2F7  85 64       STA $64
D2F9  68          PLA           Index vom Stack holen
D2FA  85 65       STA $65
D2FC  D1 5F       CMP ($5F),Y   mit Wert im Array vergleichen
D2FE  90 0E       BCC $D30E     kleiner ?
D300  D0 06       BNE $D308     größer, dann 'bad subscript'
D302  C8          INY
D303  8A          TXA
D304  D1 5F       CMP ($5F),Y   bei Gleichheit low-Byte vergleichen
D306  90 07       BCC $D30F     kleiner, dann weiter
D308  4C 45 D2    JMP $D245     gibt 'bad subscript'
D30B  4C 35 C4    JMP $C435     gibt 'out of memory'

******************************  Berechnung der Adresse eines Arrayelements
D30E  C8          INY
D30F  A5 72       LDA $72
D311  05 71       ORA $71
D313  18          CLC
D314  F0 0A       BEQ $D320
D316  20 4C D3    JSR $D34C     Multiplikation
D319  8A          TXA
D31A  65 64       ADC $64
D31C  AA          TAX
D31D  98          TYA
D31E  A4 22       LDY $22
D320  65 65       ADC $65
D322  86 71       STX $71
D324  C6 0B       DEC $0B       Anzahl der Dimensionen
D326  D0 CA       BNE $D2F2     mit nächstem Index weitermachen
D328  85 72       STA $72
```

```
D32A  A2 05       LDX #$05     Default für Variablenlänge (Real)
D32C  A5 45       LDA $45      erster Buchstabe des Namens
D32E  10 01       BPL $D331
D330  CA          DEX
D331  A5 46       LDA $46      zweiter Buchstabe des Namens
D333  10 02       BPL $D337
D335  CA          DEX
D336  CA          DEX
D337  86 28       STX $28      Länge der Variablen  2, 3 oder 5
D339  A9 00       LDA #$00
D33B  20 55 D3    JSR $D355    Offset im Array berechnen
D33E  8A          TXA
D33F  65 58       ADC $58
D341  85 47       STA $47
D343  98          TYA
D344  65 59       ADC $59
D346  85 48       STA $48
D348  A8          TAY
D349  A5 47       LDA $47
D34B  60          RTS

****************************** Hilfsroutine für Arrayberechnung
D34C  84 22       STY $22
D34E  B1 5F       LDA ($5F),Y
D350  85 28       STA $28
D352  88          DEY
D353  B1 5F       LDA ($5F),Y
D355  85 29       STA $29
D357  A9 10       LDA #$10
D359  85 5D       STA $5D
D35B  A2 00       LDX #$00
D35D  A0 00       LDY #$00
D35F  8A          TXA
D360  0A          ASL
D361  AA          TAX
D362  98          TYA
D363  2A          ROL
D364  A8          TAY
D365  B0 A4       BCS $D30B
D367  06 71       ASL $71
D369  26 72       ROL $72
D36B  90 0B       BCC $D378
D36D  18          CLC
D36E  8A          TXA
D36F  65 28       ADC $28
D371  AA          TAX
D372  98          TYA
D373  65 29       ADC $29
D375  A8          TAY
D376  B0 93       BCS $D30B
D378  C6 5D       DEC $5D
D37A  D0 E3       BNE $D35F
D37C  60          RTS

****************************** BASIC-Funktion  FRE
D37D  A5 0D       LDA $0D      Typ-Flag
D37F  F0 03       BEQ $D384    numerisch ?
D381  20 A6 D6    JSR $D6A6    FRESTR
D384  20 26 D5    JSR $D526    Garbage Collection
D387  38          SEC
```

```
D388  A5 33      LDA $33
D38A  E5 31      SBC $31      Stringanfang
D38C  A8         TAY
D38D  A5 34      LDA $34
D38F  E5 32      SBC $32      minus Variablenende
D391  A2 00      LDX #$00
D393  86 0D      STX $0D      Flag auf numerisch setzen
D395  85 62      STA $62
D397  84 63      STY $63      Ergebnis merken
D399  A2 90      LDX #$90
D39B  4C 44 DC   JMP $DC44    und nach Fließkomma wandeln

*****************************  BASIC-Funktion  POS
D39E  38         SEC          C=1
D39F  20 F0 FF   JSR $FFF0    Cursorposition holen
D3A2  A9 00      LDA #$00
D3A4  F0 EB      BEQ $D391    weiter wie oben

*****************************  Test auf Direkt-Modus
D3A6  A6 3A      LDX $3A
D3A8  E8         INX
D3A9  D0 A0      BNE $D34B    nein, dann RTS
D3AB  A2 15      LDX #$15     Nummer für 'illegal direct'
D3AD  2C         .BYTE $2C
D3AD  A2 1B      LDX #$1B     Nummer für 'undef'd function'
D3B0  4C 37 C4   JMP $C437    Fehlermeldung ausgeben

*****************************  BASIC-Befehl  DEF
D3B3  20 E1 D3   JSR $D3E1    prüft FN-Syntax
D3B6  20 A6 D3   JSR $D3A6    Direktmodus ?
D3B9  20 FA CE   JSR $CEFA    prüft auf Klammer auf
D3BC  A9 80      LDA #$80
D3BE  85 10      STA $10      sperrt Integer-Variable
D3C0  20 8B D0   JSR $D08B    sucht Variable
D3C3  20 8D CD   JSR $CD8D    prüft auf numerisch
D3C6  20 F7 CE   JSR $CEF7    prüft auf Klammer zu
D3C9  A9 B2      LDA #$B2     '=' - Kode
D3CB  20 FF CE   JSR $CEFF    prüft auf Kode
D3CE  48         PHA
D3CF  A5 48      LDA $48
D3D1  48         PHA          FN-Variablenadresse auf Stack
D3D2  A5 47      LDA $47
D3D4  48         PHA
D3D5  A5 7B      LDA $7B
D3D7  48         PHA          Programmzeiger auf Stack
D3D8  A5 7A      LDA $7A
D3DA  48         PHA
D3DB  20 F8 C8   JSR $C8F8    Programmzeiger auf nächstes Statement
D3DE  4C 4F D4   JMP $D44F    FN-Variable vom Stack holen

*****************************  prüft FN-Syntax
D3E1  A9 A5      LDA #$A5     'FN' -Kode
D3E3  20 FF CE   JSR $CEFF    prüft auf Kode
D3E6  09 80      ORA #$80
D3E8  85 10      STA $10      sperrt Integer-Variable
D3EA  20 92 D0   JSR $D092    sucht Variable
D3ED  85 4E      STA $4E
D3EF  84 4F      STY $4F      FN-Variablenzeiger setzen
D3F1  4C 8D CD   JMP $CD8D    prüft auf numerisch
```

```
*****************************    BASIC-Funktion  FN
D3F4   20 E1 D3    JSR $D3E1     prüft FN-Syntax
D3F7   A5 4F       LDA $4F
D3F9   48          PHA           FN-Variablenzeiger auf Stack
D3FA   A5 4E       LDA $4E
D3FC   48          PHA
D3FD   20 F1 CE    JSR $CEF1     holt Ausdruck in Klammern
D400   20 8D CD    JSR $CD8D     prüft auf numerisch
D403   68          PLA
D404   85 4E       STA $4E
D406   68          PLA           FN-Variablenzeiger zurückholen
D407   85 4F       STA $4F
D409   A0 02       LDY #$02
D40B   B1 4E       LDA ($4E),Y
D40D   85 47       STA $47
D40F   AA          TAX
D410   C8          INY
D411   B1 4E       LDA ($4E),Y
D413   F0 99       BEQ $D3AE     gibt 'undef'd function'
D415   85 48       STA $48
D417   C8          INY
D418   B1 47       LDA ($47),Y
D41A   48          PHA
D41B   88          DEY
D41C   10 FA       BPL $D418
D41E   A4 48       LDY $48
D420   20 D4 DB    JSR $DBD4     FAC nach FN-Variable übertragen
D423   A5 7B       LDA $7B
D425   48          PHA           Programmzeiger auf FN-Ausdruck
D426   A5 7A       LDA $7A
D428   48          PHA
D429   B1 4E       LDA ($4E),Y
D42B   85 7A       STA $7A
D42D   C8          INY
D42E   B1 4E       LDA ($4E),Y
D430   85 7B       STA $7B
D432   A5 48       LDA $48
D434   48          PHA
D435   A5 47       LDA $47
D437   48          PHA
D438   20 8A CD    JSR $CD8A      FRMNUM numerischen Ausdruck holen
D43B   68          PLA
D43C   85 4E       STA $4E
D43E   68          PLA
D43F   85 4F       STA $4F
D441   20 79 00    JSR $0079     CHRGOT  laufendes Zeichen holen
D444   F0 03       BEQ $D449     keine weiteren Zeichen ?
D446   4C 08 CF    JMP $CF08     gibt 'syntax error'
D449   68          PLA
D44A   85 7A       STA $7A
D44C   68          PLA           Programmzeiger zurückholen
D44D   85 7B       STA $7B
D44F   A0 00       LDY #$00
D451   68          PLA
D452   91 4E       STA ($4E),Y
D454   68          PLA
D455   C8          INY
D456   91 4E       STA ($4E),Y
D458   68          PLA           FN-Variable zurückholen
D459   C8          INY
```

```
D45A  91 4E      STA ($4E),Y
D45C  68         PLA
D45D  C8         INY
D45E  91 4E      STA ($4E),Y
D460  68         PLA
D461  C8         INY
D462  91 4E      STA ($4E),Y
D464  60         RTS

*****************************   BASIC-Funktion  STR$
D465  20 8D CD   JSR $CD8D      prüft auf numerisch
D468  A0 00      LDY #$00
D46A  20 DF DD   JSR $DDDF       FAC nach ASCII umwandeln
D46D  68         PLA
D46E  68         PLA
D46F  A9 FF      LDA #$FF
D471  A0 00      LDY #$00       Startadresse auf $FF
D473  F0 12      BEQ $D487

*****************************   Stringverwaltung
D475  A6 64      LDX $64
D477  A4 65      LDY $65
D479  86 50      STX $50        Zeiger auf Stringdescriptor
D47B  84 51      STY $51
D47D  20 F4 D4   JSR $D4F4      prüft auf Speicherplatz, setzt Stringzeiger
D480  86 62      STX $62
D482  84 63      STY $63
D484  85 61      STA $61
D486  60         RTS

D487  A2 22      LDX #$22       '"'
D489  86 07      STX $07
D48B  86 08      STX $08
D48D  85 6F      STA $6F        Startadresse des Strings
D48F  84 70      STY $70
D491  85 62      STA $62
D493  84 63      STY $63
D495  A0 FF      LDY #$FF
D497  C8         INY            Zeiger erhöhen
D498  B1 6F      LDA ($6F),Y    nächstes Zeichen des Strings
D49A  F0 0C      BEQ $D4A8      Endekennzeichen ?
D49C  C5 07      CMP $07
D49E  F0 04      BEQ $D4A4
D4A0  C5 08      CMP $08
D4A2  D0 F3      BNE $D497
D4A4  C9 22      CMP #$22       '"'
D4A6  F0 01      BEQ $D4A9
D4A8  18         CLC
D4A9  84 61      STY $61        Länge des Strings
D4AB  98         TYA
D4AC  65 6F      ADC $6F
D4AE  85 71      STA $71
D4B0  A6 70      LDX $70
D4B2  90 01      BCC $D4B5
D4B4  E8         INX
D4B5  86 72      STX $72        Endadresse high + 1
D4B7  A5 70      LDA $70        Startadresse high
D4B9  F0 04      BEQ $D4BF      null ?
D4BB  C9 02      CMP #$02       zwei ?
D4BD  D0 0B      BNE $D4CA      nein
```

```
D4BF  98          TYA
D4C0  20 75 D4    JSR $D475     Stringdescriptor holen, Länge A, Adresse X/Y
D4C3  A6 6F       LDX $6F
D4C5  A4 70       LDY $70       Startadresse holen
D4C7  20 88 D6    JSR $D688     String in Stringbereich kopieren
D4CA  A6 16       LDX $16       Stringdescriptor zeigen
D4CC  E0 22       CPX #$22      Stringstack voll ?
D4CE  D0 05       BNE $D4D5     nein
D4D0  A2 19       LDX #$19      Nummer für 'formula too complex'
D4D2  4C 37 C4    JMP $C437     Fehlermeldung ausgeben
D4D5  A5 61       LDA $61
D4D7  95 00       STA $00,X     Stringlänge
D4D9  A5 62       LDA $62
D4DB  95 01       STA $01,X
D4DD  A5 63       LDA $63       und Adresse in Stringstack bringen
D4DF  95 02       STA $02,X
D4E1  A0 00       LDY #$00
D4E3  86 64       STX $64
D4E5  84 65       STY $65       Zeiger jetzt auf Descriptor
D4E7  84 70       STY $70
D4E9  88          DEY
D4EA  84 0D       STY $0D       Stringflag setzen $FF
D4EC  86 17       STX $17       Index des letzten Stringdescriptors
D4EE  E8          INX
D4EF  E8          INX           um drei erhöhen
D4F0  E8          INX
D4F1  86 16       STX $16       als neuen Index merken
D4F3  60          RTS
D4F4  46 0F       LSR $0F       Flag für Garbage Collect rücksetzen
D4F6  48          PHA
D4F7  49 FF       EOR #$FF
D4F9  38          SEC
D4FA  65 33       ADC $33
D4FC  A4 34       LDY $34
D4FE  B0 01       BCS $D501
D500  88          DEY
D501  C4 32       CPY $32
D503  90 11       BCC $D516
D505  D0 04       BNE $D50B
D507  C5 31       CMP $31
D509  90 0B       BCC $D516
D50B  85 33       STA $33
D50D  84 34       STY $34
D50F  85 35       STA $35
D511  84 36       STY $36
D513  AA          TAX
D514  68          PLA
D515  60          RTS

D516  A2 10       LDX #$10      Nummer für 'out of memory'
D518  A5 0F       LDA $0F       Flag für Garbage Collect
D51A  30 B6       BMI $D4D2     bereits durchgeführt, dann 'out of memory'
D51C  20 26 D5    JSR $D526     Garbage Collection
D51F  A9 80       LDA #$80      Flag setzen
D521  85 0F       STA $0F
D523  68          PLA
D524  D0 D0       BNE $D4F6

******************************  Garbage Collect
D526  A6 37       LDX $37
```

```
D528  A5 38     LDA $38
D52A  86 33     STX $33
D52C  85 34     STA $34
D52E  A0 00     LDY #$00
D530  84 4F     STY $4F
D532  84 4E     STY $4E
D534  A5 31     LDA $31
D536  A6 32     LDX $32
D538  85 5F     STA $5F
D53A  86 60     STX $60
D53C  A9 19     LDA #$19
D53E  A2 00     LDX #$00
D540  85 22     STA $22
D542  86 23     STX $23
D544  C5 16     CMP $16
D546  F0 05     BEQ $D54D
D548  20 C7 D5  JSR $D5C7
D54B  F0 F7     BEQ $D544
D54D  A9 07     LDA #$07
D54F  85 53     STA $53
D551  A5 2D     LDA $2D
D553  A6 2E     LDX $2E
D555  85 22     STA $22
D557  86 23     STX $23
D559  E4 30     CPX $30
D55B  D0 04     BNE $D561
D55D  C5 2F     CMP $2F
D55F  F0 05     BEQ $D566
D561  20 BD D5  JSR $D5BD
D564  F0 F3     BEQ $D559
D566  85 58     STA $58
D568  86 59     STX $59
D56A  A9 03     LDA #$03
D56C  85 53     STA $53
D56E  A5 58     LDA $58
D570  A6 59     LDX $59
D572  E4 32     CPX $32
D574  D0 07     BNE $D57D
D576  C5 31     CMP $31
D578  D0 03     BNE $D57D
D57A  4C 06 D6  JMP $D606
D57D  85 22     STA $22
D57F  86 23     STX $23
D581  A0 00     LDY #$00
D583  B1 22     LDA ($22),Y
D585  AA        TAX
D586  C8        INY
D587  B1 22     LDA ($22),Y
D589  08        PHP
D58A  C8        INY
D58B  B1 22     LDA ($22),Y
D58D  65 58     ADC $58
D58F  85 58     STA $58
D591  C8        INY
D592  B1 22     LDA ($22),Y
D594  65 59     ADC $59
D596  85 59     STA $59
D598  28        PLP
D599  10 D3     BPL $D56E
D59B  8A        TXA
```

```
D59C   30 D0        BMI $D56E
D59E   C8           INY
D59F   B1 22        LDA ($22),Y
D5A1   A0 00        LDY #$00
D5A3   0A           ASL
D5A4   69 05        ADC #$05
D5A6   65 22        ADC $22
D5A8   85 22        STA $22
D5AA   90 02        BCC $D5AE
D5AC   E6 23        INC $23
D5AE   A6 23        LDX $23
D5B0   E4 59        CPX $59
D5B2   D0 04        BNE $D5B8
D5B4   C5 58        CMP $58
D5B6   F0 BA        BEQ $D572
D5B8   20 C7 D5     JSR $D5C7
D5BB   F0 F3        BEQ $D5B0
D5BD   B1 22        LDA ($22),Y
D5BF   30 35        BMI $D5F6
D5C1   C8           INY
D5C2   B1 22        LDA ($22),Y
D5C4   10 30        BPL $D5F6
D5C6   C8           INY
D5C7   B1 22        LDA ($22),Y
D5C9   F0 2B        BEQ $D5F6
D5CB   C8           INY
D5CC   B1 22        LDA ($22),Y
D5CE   AA           TAX
D5CF   C8           INY
D5D0   B1 22        LDA ($22),Y
D5D2   C5 34        CMP $34
D5D4   90 06        BCC $D5DC
D5D6   D0 1E        BNE $D5F6
D5D8   E4 33        CPX $33
D5DA   B0 1A        BCS $D5F6
D5DC   C5 60        CMP $60
D5DE   90 16        BCC $D5F6
D5E0   D0 04        BNE $D5E6
D5E2   E4 5F        CPX $5F
D5E4   90 10        BCC $D5F6
D5E6   86 5F        STX $5F
D5E8   85 60        STA $60
D5EA   A5 22        LDA $22
D5EC   A6 23        LDX $23
D5EE   85 4E        STA $4E
D5F0   86 4F        STX $4F
D5F2   A5 53        LDA $53
D5F4   85 55        STA $55
D5F6   A5 53        LDA $53
D5F8   18           CLC
D5F9   65 22        ADC $22
D5FB   85 22        STA $22
D5FD   90 02        BCC $D601
D5FF   E6 23        INC $23
D601   A6 23        LDX $23
D603   A0 00        LDY #$00
D605   60           RTS
D606   A5 4F        LDA $4F
D608   05 4E        ORA $4E
D60A   F0 F5        BEQ $D601
```

```
D60C  A5 55       LDA $55
D60E  29 04       AND #$04
D610  4A          LSR
D611  A8          TAY
D612  85 55       STA $55
D614  B1 4E       LDA ($4E),Y
D616  65 5F       ADC $5F
D618  85 5A       STA $5A
D61A  A5 60       LDA $60
D61C  69 00       ADC #$00
D61E  85 5B       STA $5B
D620  A5 33       LDA $33
D622  A6 34       LDX $34
D624  85 58       STA $58
D626  86 59       STX $59
D628  20 BF C3    JSR $C3BF
D62B  A4 55       LDY $55
D62D  C8          INY
D62E  A5 58       LDA $58
D630  91 4E       STA ($4E),Y
D632  AA          TAX
D633  E6 59       INC $59
D635  A5 59       LDA $59
D637  C8          INY
D638  91 4E       STA ($4E),Y
D63A  4C 2A D5    JMP $D52A

******************************    Stringverknüpfung +
D63D  A5 65       LDA $65
D63F  48          PHA             Descriptor des ersten Strings merken
D640  A5 64       LDA $64
D642  48          PHA
D643  20 83 CE    JSR $CE83       Ausdruck holen
D646  20 8F CD    JSR $CD8F       prüft auf Stringvariable
D649  68          PLA
D64A  85 6F       STA $6F
D64C  68          PLA             Descriptor zurückholen
D64D  85 70       STA $70
D64F  A0 00       LDY #$00
D651  B1 6F       LDA ($6F),Y     Länge des ersten Strings
D653  18          CLC
D654  71 64       ADC ($64),Y     plus Länge des zweiten Strings
D656  90 05       BCC $D65D       kleiner als 256 ?
D658  A2 17       LDX #$17        Nummer für 'string too long'
D65A  4C 37 C4    JMP $C437       Fehlermeldung ausgeben
D65D  20 75 D4    JSR $D475       Platz für verknüpften String reservieren
D660  20 7A D6    JSR $D67A       ersten String dorthin übertragen
D663  A5 50       LDA $50
D665  A4 51       LDY $51         Zeiger auf zweiten Stringdescriptor
D667  20 AA D6    JSR $D6AA       FRESTR
D66A  20 8C D6    JSR $D68C       zweiten String an ersten anhängen
D66D  A5 6F       LDA $6F
D66F  A4 70       LDY $70
D671  20 AA D6    JSR $D6AA       FRESTR
D674  20 CA D4    JSR $D4CA       Descriptor in Stringstack
D677  4C B8 CD    JMP $CDB8       zurück zur Formelauswertung

D67A  A0 00       LDY #$00
D67C  B1 6F       LDA ($6F),Y     Stringlänge merken
D67E  48          PHA
```

```
D67F  C8          INY
D680  B1 6F       LDA ($6F),Y   Stringadresse low
D682  AA          TAX
D683  C8          INY
D684  B1 6F       LDA ($6F),Y   Stringadresse high
D686  A8          TAY
D687  68          PLA           Stringlänge
D688  86 22       STX $22
D68A  84 23       STY $23       Zeiger auf String
D68C  A8          TAY
D68D  F0 0A       BEQ $D699     Länge null, dann fertig
D68F  48          PHA
D690  88          DEY
D691  B1 22       LDA ($22),Y   String übertragen
D693  91 35       STA ($35),Y
D695  98          TYA
D696  D0 F8       BNE $D690
D698  68          PLA
D699  18          CLC
D69A  65 35       ADC $35
D69C  85 35       STA $35       Zeiger plus Stringlänge
D69E  90 02       BCC $D6A2
D6A0  E6 36       INC $36
D6A2  60          RTS

******************************  Stringverwaltung  FRESTR
D6A3  20 8F CD    JSR $CD8F     prüft auf Stringvariable
D6A6  A5 64       LDA $64
D6A8  A4 65       LDY $65       Zeiger auf Stringdescriptor
D6AA  85 22       STA $22
D6AC  84 23       STY $23
D6AE  20 DB D6    JSR $D6DB     Descriptor vom Stringstack entfernen
D6B1  08          PHP
D6B2  A0 00       LDY #$00
D6B4  B1 22       LDA ($22),Y
D6B6  48          PHA
D6B7  C8          INY
D6B8  B1 22       LDA ($22),Y
D6BA  AA          TAX
D6BB  C8          INY
D6BC  B1 22       LDA ($22),Y
D6BE  A8          TAY
D6BF  68          PLA
D6C0  28          PLP
D6C1  D0 13       BNE $D6D6     String war nicht im Stringstack
D6C3  C4 34       CPY $34
D6C5  D0 0F       BNE $D6D6
D6C7  E4 33       CPX $33
D6C9  D0 0B       BNE $D6D6
D6CB  48          PHA
D6CC  18          CLC
D6CD  65 33       ADC $33
D6CF  85 33       STA $33
D6D1  90 02       BCC $D6D5
D6D3  E6 34       INC $34
D6D5  68          PLA
D6D6  86 22       STX $22
D6D8  84 23       STY $23
D6DA  60          RTS
D6DB  C4 18       CPY $18
```

```
D6DD  D0 0C      BNE $D6EB
D6DF  C5 17      CMP $17
D6E1  D0 08      BNE $D6EB
D6E3  85 16      STA $16
D6E5  E9 03      SBC #$03
D6E7  85 17      STA $17
D6E9  A0 00      LDY #$00
D6EB  60         RTS

******************************   BASIC-Funktion  CHR$
D6EC  20 A1 D7   JSR $D7A1       holt Byte-Wert
D6EF  8A         TXA
D6F0  48         PHA
D6F1  A9 01      LDA #$01        Länge des Strings gleich eins
D6F3  20 7D D4   JSR $D47D       Platz für String reservieren
D6F6  68         PLA             ASCII-Kode zurückholen
D6F7  A0 00      LDY #$00
D6F9  91 62      STA ($62),Y     als Stringzeichen speichern
D6FB  68         PLA
D6FC  68         PLA
D6FD  4C CA D4   JMP $D4CA       Stringdescriptor in Tabelle bringen

******************************   Basic-Funktion  LEFT$
D700  20 61 D7   JSR $D761       Stringparamter von Stack holen
D703  D1 50      CMP ($50),Y     Länge mit LEFT$-Parameter vergleichen
D705  98         TYA
D706  90 04      BCC $D70C       LEFT$-Parameter kleiner als Länge ?
D708  B1 50      LDA ($50),Y     Stringlänge
D70A  AA         TAX
D70B  98         TYA
D70C  48         PHA
D70D  8A         TXA
D70E  48         PHA
D70F  20 7D D4   JSR $D47D       Platz für neuen String reservieren
D712  A5 50      LDA $50
D714  A4 51      LDY $51         Zeiger auf Descriptor
D716  20 AA D6   JSR $D6AA       FRESTR
D719  68         PLA
D71A  A8         TAY
D71B  68         PLA
D71C  18         CLC
D71D  65 22      ADC $22
D71F  85 22      STA $22
D721  90 02      BCC $D725
D723  E6 23      INC $23
D725  98         TYA
D726  20 8C D6   JSR $D68C       neuen String in Stringbereich kopieren
D729  4C CA D4   JMP $D4CA       Decriptor in Stringstack bringen

******************************   BASIC-Funktion  RIGHT$
D72C  20 61 D7   JSR $D761       Stringparameter vom Stack holen
D72F  18         CLC
D730  F1 50      SBC ($50),Y     Nummer des ersten Elements im alten String
D732  49 FF      EOR #$FF
D734  4C 06 D7   JMP $D706       weiter wie LEFT$

******************************   BASIC-Funktion  MID$
D737  A9 FF      LDA #$FF
D739  85 65      STA $65
D73B  20 79 00   JSR $0079       CHRGOT  laufendes Zeichen holen
```

```
D73E  C9 29     CMP #$29      ')'  Klammer zu
D740  F0 06     BEQ $D748
D742  20 FD CE  JSR $CEFD     CHKCOM  prüft auf Komma
D745  20 9E D7  JSR $D79E     holt Byte-Wert
D748  20 61 D7  JSR $D761     holt Stringadresse und 1. Parameter
D74B  F0 4B     BEQ $D798     1. Parameter null, dann 'illegal quantity'
D74D  CA        DEX
D74E  8A        TXA
D74F  48        PHA           Nummer des ersten Elements im alten String
D750  18        CLC
D751  A2 00     LDX #$00
D753  F1 50     SBC ($50),Y   Länge des alten Strings
D755  B0 B6     BCS $D70D     kleiner erster MID-Parameter ?
D757  49 FF     EOR #$FF
D759  C5 65     CMP $65
D75B  90 B1     BCC $D70E
D75D  A5 65     LDA $65
D75F  B0 AD     BCS $D70E     unbedingter Sprung

D761  20 F7 CE  JSR $CEF7     prüft auf Klammer zu
D764  68        PLA
D765  A8        TAY
D766  68        PLA
D767  85 55     STA $55
D769  68        PLA
D76A  68        PLA
D76B  68        PLA
D76C  AA        TAX
D76D  68        PLA
D76E  85 50     STA $50
D770  68        PLA
D771  85 51     STA $51
D773  A5 55     LDA $55
D775  48        PHA
D776  98        TYA
D777  48        PHA
D778  A0 00     LDY #$00
D77A  8A        TXA
D77B  60        RTS

******************************  BASIC-Funktion  LEN
D77C  20 82 D7  JSR $D782     FRESTR, Stringlänge holen
D77F  4C A2 D3  JMP $D3A2     Byte-Wert nach Fließkomma wandeln

******************************  Stringparameter holen
D782  20 A3 D6  JSR $D6A3     String holen, Länge in A
D785  A2 00     LDX #$00
D787  86 0D     STX $0D       Typflag auf numerisch
D789  A8        TAY
D78A  60        RTS

******************************  BASIC-Funktion  ASC
D78B  20 82 D7  JSR $D782     String holen Adresse nach $22/$23, Länge nach Y
D78E  F0 08     BEQ $D798     Länge gleich Null, 'illegal qunatity'
D790  A0 00     LDY #$00
D792  B1 22     LDA ($22),Y   erstes Zeichen holen
D794  A8        TAY           ASCII-Kode
D795  4C A2 D3  JMP $D3A2     nach Fließkomma wandeln

D798  4C 48 D2  JMP $D248     gibt 'illegal quantity'
```

```
******************************      holt Byte-Wert (0-255) nach X
D79B  20 73 00    JSR $0073         CHRGET  nächstes Zeichen holen
D79E  20 8A CD    JSR $CD8A         FRMNUM  numerischen Ausdruck holen
D7A1  20 B8 D1    JSR $D1B8         prüft auf Bereich und wandelt nach Integer
D7A4  A6 64       LDX $64           high Byte
D7A6  D0 F0       BNE $D798         ungleich null, dann Wert zu groß
D7A8  A6 65       LDX $65
D7AA  4C 79 00    JMP $0079         CHRGOT  letzes Zeichen holen

******************************      BASIC-Funktion  VAL
D7AD  20 82 D7    JSR $D782         Stringadresse und Länge holen
D7B0  D0 03       BNE $D7B5         Länge ungleich Null ?
D7B2  4C F7 D8    JMP $D8F7         Null in FAC
D7B5  A6 7A       LDX $7A
D7B7  A4 7B       LDY $7B           Programmzeiger retten
D7B9  86 71       STX $71
D7BB  84 72       STY $72
D7BD  A6 22       LDX $22
D7BF  86 7A       STX $7A
D7C1  18          CLC
D7C2  65 22       ADC $22
D7C4  85 24       STA $24
D7C6  A6 23       LDX $23
D7C8  86 7B       STX $7B
D7CA  90 01       BCC $D7CD
D7CC  E8          INX
D7CD  86 25       STX $25
D7CF  A0 00       LDY #$00
D7D1  B1 24       LDA ($24),Y
D7D3  48          PHA
D7D4  98          TYA
D7D5  91 24       STA ($24),Y
D7D7  20 79 00    JSR $0079         CHRGOT  laufendes Zeichen holen
D7DA  20 F3 DC    JSR $DCF3         String in Fließkommazahl umwandeln
D7DD  68          PLA
D7DE  A0 00       LDY #$00
D7E0  91 24       STA ($24),Y
D7E2  A6 71       LDX $71
D7E4  A4 72       LDY $72
D7E6  86 7A       STX $7A           Programmzeiger zurückholen
D7E8  84 7B       STY $7B
D7EA  60          RTS

******************************      GETADR und GETBYT holt 16-Bit und 8-Bit Wert
D7EB  20 8A CD    JSR $CD8A         FRMNUM  holt numerischen Wert
D7EE  20 F7 D7    JSR $D7F7         FAC in Adressformat wandeln
D7F1  20 FD CE    JSR $CEFD         CHKCOM prüft auf Komma
D7F4  4C 9E D7    JMP $D79E         holt Byte-Wert nach X

******************************      FAC nach Adressformat umwandeln
D7F7  A5 66       LDA $66           Vorzeichen
D7F9  30 9D       BMI $D798         negativ, dann 'illegal quantity'
D7FB  A5 61       LDA $61           Exponent
D7FD  C9 91       CMP #$91          Zahl mit 65536 vergleichen
D7FF  B0 97       BCS $D798         größer oder gleich, 'illegal quantity'
D801  20 9B DC    JSR $DC9B         in Adressformat wandeln
D804  A5 64       LDA $64
D806  A4 65       LDY $65           Wert nach
D808  84 14       STY $14           $14/$15 speichern
D80A  85 15       STA $15
```

```
D80C  60          RTS

******************************   BASIC-Funktion  PEEK
D80D  A5 15       LDA $15
D80F  48          PHA            Adresse in $14/$15 sichern
D810  A5 14       LDA $14
D812  48          PHA
D813  20 F7 D7    JSR $D7F7      FAC in Adressformat wandeln
D816  A0 00       LDY #$00
D818  B1 14       LDA ($14),Y    Peek-Wert holen
D81A  A8          TAY
D81B  68          PLA
D81C  85 14       STA $14
D81E  68          PLA            Adresse zurückholen
D81F  85 15       STA $15
D821  4C A2 D3    JMP $D3A2      Y in Fließkommaformat wandeln

******************************   BASIC-Befehl POKE
D824  20 EB D7    JSR $D7EB      Poke-Adresse und Wert holen
D827  8A          TXA            Wert in Akku
D828  A0 00       LDY #$00
D82A  91 14       STA ($14),Y    und speichern
D82C  60          RTS

******************************   BASIC-Befehl  WAIT
D82D  20 EB D7    JSR $D7EB      Adresse und Wert holen
D830  86 49       STX $49
D832  A2 00       LDX #$00       Default für dritten Parameter
D834  20 79 00    JSR $0079      CHRGOT  laufendes Zeichen holen
D837  F0 03       BEQ $D83C      kein dritter Parameter ?
D839  20 F1 D7    JSR $D7F1      prüft auf Komma und holt Parameter
D83C  86 4A       STX $4A
D83E  A0 00       LDY #$00
D840  B1 14       LDA ($14),Y    Wait-Adresse
D842  45 4A       EOR $4A
D844  25 49       AND $49        logisch verknüpfen
D846  F0 F8       BEQ $D840      weiter warten
D848  60          RTS

******************************   FAC = FAC +0.5
D849  A9 11       LDA #$11
D84B  A0 DF       LDY #$DF       Zeiger auf Konstante 0.5
D84D  4C 67 D8    JMP $D867      FAC = FAC + Konstante (A/Y)

******************************   Minus  FAC = Konstante (A/Y) - FAC
D850  20 8C DA    JSR $DA8C      Konstante (A/Y) nach ARG

******************************   Minus  FAC = ARG - FAC
D853  A5 66       LDA $66
D855  49 FF       EOR #$FF       Vorzeichen umdrehen
D857  85 66       STA $66
D859  45 6E       EOR $6E
D85B  85 6F       STA $6F
D85D  A5 61       LDA $61
D85F  4C 6A D8    JMP $D86A      FAC = FAC + ARG

******************************
D862  20 99 D9    JSR $D999      Exponenten von FAC und ARG angleichen
D865  90 3C       BCC $D8A3
```

```
******************************    Plus  FAC = Konstante (A/Y) + FAC
D867   20 8C DA    JSR $DA8C      Konstante (A/Y) nach ARG

******************************    Plus  FAC = ARG + FAC
D86A   D0 03       BNE $D86F      FAC ungleich Null ?
D86C   4C FC DB    JMP $DBFC      nein, dann FAC = ARG
D86F   A6 70       LDX $70
D871   86 56       STX $56
D873   A2 69       LDX #$69
D875   A5 69       LDA $69
D877   A8          TAY
D878   F0 CE       BEQ $D848
D87A   38          SEC
D87B   E5 61       SBC $61
D87D   F0 24       BEQ $D8A3
D87F   90 12       BCC $D893
D881   84 61       STY $61
D883   A4 6E       LDY $6E
D885   84 66       STY $66
D887   49 FF       EOR #$FF
D889   69 00       ADC #$00
D88B   A0 00       LDY #$00
D88D   84 56       STY $56
D88F   A2 61       LDX #$61
D891   D0 04       BNE $D897
D893   A0 00       LDY #$00
D895   84 70       STY $70
D897   C9 F9       CMP #$F9
D899   30 C7       BMI $D862
D89B   A8          TAY
D89C   A5 70       LDA $70
D89E   56 01       LSR $01,X
D8A0   20 B0 D9    JSR $D9B0
D8A3   24 6F       BIT $6F
D8A5   10 57       BPL $D8FE
D8A7   A0 61       LDY #$61
D8A9   E0 69       CPX #$69
D8AB   F0 02       BEQ $D8AF
D8AD   A0 69       LDY #$69
D8AF   38          SEC
D8B0   49 FF       EOR #$FF
D8B2   65 56       ADC $56
D8B4   85 70       STA $70
D8B6   B9 04 00    LDA $0004,Y
D8B9   F5 04       SBC $04,X
D8BB   85 65       STA $65
D8BD   B9 03 00    LDA $0003,Y
D8C0   F5 03       SBC $03,X
D8C2   85 64       STA $64
D8C4   B9 02 00    LDA $0002,Y
D8C7   F5 02       SBC $02,X
D8C9   85 63       STA $63
D8CB   B9 01 00    LDA $0001,Y
D8CE   F5 01       SBC $01,X
D8D0   85 62       STA $62
D8D2   B0 03       BCS $D8D7
D8D4   20 47 D9    JSR $D947
D8D7   A0 00       LDY #$00
D8D9   98          TYA
D8DA   18          CLC
```

```
D8DB  A6 62       LDX $62
D8DD  D0 4A       BNE $D929
D8DF  A6 63       LDX $63
D8E1  86 62       STX $62
D8E3  A6 64       LDX $64
D8E5  86 63       STX $63
D8E7  A6 65       LDX $65
D8E9  86 64       STX $64
D8EB  A6 70       LDX $70
D8ED  86 65       STX $65
D8EF  84 70       STY $70
D8F1  69 08       ADC #$08
D8F3  C9 20       CMP #$20
D8F5  D0 E4       BNE $D8DB
D8F7  A9 00       LDA #$00
D8F9  85 61       STA $61
D8FB  85 66       STA $66
D8FD  60          RTS
D8FE  65 56       ADC $56
D900  85 70       STA $70
D902  A5 65       LDA $65
D904  65 6D       ADC $6D
D906  85 65       STA $65
D908  A5 64       LDA $64
D90A  65 6C       ADC $6C
D90C  85 64       STA $64
D90E  A5 63       LDA $63
D910  65 6B       ADC $6B
D912  85 63       STA $63
D914  A5 62       LDA $62
D916  65 6A       ADC $6A
D918  85 62       STA $62
D91A  4C 36 D9    JMP $D936
D91D  69 01       ADC #$01
D91F  06 70       ASL $70
D921  26 65       ROL $65
D923  26 64       ROL $64
D925  26 63       ROL $63
D927  26 62       ROL $62
D929  10 F2       BPL $D91D
D92B  38          SEC
D92C  E5 61       SBC $61
D92E  B0 C7       BCS $D8F7
D930  49 FF       EOR #$FF
D932  69 01       ADC #$01
D934  85 61       STA $61
D936  90 0E       BCC $D946
D938  E6 61       INC $61
D93A  F0 42       BEQ $D97E
D93C  66 62       ROR $62
D93E  66 63       ROR $63
D940  66 64       ROR $64
D942  66 65       ROR $65
D944  66 70       ROR $70
D946  60          RTS

******************************   Mantisse von FAC invertieren
D947  A5 66       LDA $66
D949  49 FF       EOR #$FF
D94B  85 66       STA $66
```

```
D94D  A5 62     LDA $62
D94F  49 FF     EOR #$FF
D951  85 62     STA $62
D953  A5 63     LDA $63
D955  49 FF     EOR #$FF
D957  85 63     STA $63
D959  A5 64     LDA $64
D95B  49 FF     EOR #$FF
D95D  85 64     STA $64
D95F  A5 65     LDA $65
D961  49 FF     EOR #$FF
D963  85 65     STA $65
D965  A5 70     LDA $70
D967  49 FF     EOR #$FF
D969  85 70     STA $70
D96B  E6 70     INC $70
D96D  D0 0E     BNE $D97D
D96F  E6 65     INC $65
D971  D0 0A     BNE $D97D
D973  E6 64     INC $64
D975  D0 06     BNE $D97D
D977  E6 63     INC $63
D979  D0 02     BNE $D97D
D97B  E6 62     INC $62
D97D  60        RTS
D97E  A2 0F     LDX #$0F      Nummer für 'overflow'
D980  4C 37 C4  JMP $C437     Fehlermeldung ausgeben

******************************   Rechtsverschieben eines Registers
D983  A2 25     LDX #$25
D985  B4 04     LDY $04
D987  84 70     STY $70
D989  B4 03     LDY $03
D98B  94 04     STY $04
D98D  B4 02     LDY $02
D98F  94 03     STY $03
D991  B4 01     LDY $01
D993  94 02     STY $02
D995  A4 68     LDY $68
D997  94 01     STY $01
D999  69 08     ADC #$08
D99B  30 E8     BMI $D985
D99D  F0 E6     BEQ $D985
D99F  E9 08     SBC #$08
D9A1  A8        TAY
D9A2  A5 70     LDA $70
D9A4  B0 14     BCS $D9BA
D9A6  16 01     ASL $01,X
D9A8  90 02     BCC $D9AC
D9AA  F6 01     INC $01,X
D9AC  76 01     ROR $01,X
D9AE  76 01     ROR $01,X
D9B0  76 02     ROR $02,X
D9B2  76 03     ROR $03,X
D9B4  76 04     ROR $04,X
D9B6  6A        ROR
D9B7  C8        INY
D9B8  D0 EC     BNE $D9A6
D9BA  18        CLC
D9BB  60        RTS
```

```
******************************  Konstanten für LOG
D9BC  81 00 00 00 00            1
D9C1  03                        3 = Polynomgrad, dann 4 Koeffizienten
D9C2  7F 5E 56 CB 79            .434255942
D9C7  80 13 9B 0B 64            .576584541
D9CC  80 76 38 93 16            .961800759
D9D1  82 38 AA 3B 20            2.88539007
D9D6  80 35 04 F3 34            .707106781  =  1/SQR(2)
D9DB  81 35 04 F3 34            1.41421356  =  SQR(2)
D9E0  80 80 00 00 00            -.5
D9E5  80 31 72 17 F8            .693147181  =  LOG(2)

******************************  BASIC-Funktion  LOG
D9EA  20 2B DC    JSR $DC2B     Vorzeichen von FAC holen
D9ED  F0 02       BEQ $D9F1     null, dann fertig
D9EF  10 03       BPL $D9F4     positiv, ok
D9F1  4C 48 D2    JMP $D248     negativ, 'illegal qunatity'
D9F4  A5 61       LDA $61       Exponent
D9F6  E9 7F       SBC #$7F      normalisieren
D9F8  48          PHA           und merken
D9F9  A9 80       LDA #$80      Zahl in Bereich 0.5 bis 1 bringen
D9FB  85 61       STA $61
D9FD  A9 D6       LDA #$D6
D9FF  A0 D9       LDY #$D9      Zeiger auf Konstante  1/SQR(2)
DA01  20 67 D8    JSR $D867     zu FAC addieren
DA04  A9 DB       LDA #$DB
DA06  A0 D9       LDY #$D9      Zeiger auf Konstante  SQR(2)
DA08  20 0F DB    JSR $DB0F     durch FAC dividieren
DA0B  A9 BC       LDA #$BC
DA0D  A0 D9       LDY #$D9      Zeiger auf Konstante 1
DA0F  20 50 D8    JSR $D850     1 minus FAC
DA12  A9 C1       LDA #$C1
DA14  A0 D9       LDY #$D9      Zeiger auf Polynomkoeffizienten
DA16  20 40 E0    JSR $E040     Polynomberechnung
DA19  A9 E0       LDA #$E0
DA1B  A0 D9       LDY #$D9      Zeiger auf Konstante -0.5
DA1D  20 67 D8    JSR $D867     zu FAC addieren
DA20  68          PLA           Exponent zurückholen
DA21  20 7E DD    JSR $DD7E     FAC = FAC + FAC
DA24  A9 E5       LDA #$E5
DA26  A0 D9       LDY #$D9      Zeiger auf Konstante  LOG(2)

******************************  Multiplikation  FAC = Konstante (A/Y) * FAC
DA28  20 8C DA    JSR $DA8C     Konstante (A/Y) nach FAC holen

******************************  Multiplikation  FAC = ARG * FAC
DA2B  D0 03       BNE $DA30     nicht null ?
DA2D  4C 8B DA    JMP $DA8B     gibt RTS
DA30  20 B7 DA    JSR $DAB7     Exponent berechnen
DA33  A9 00       LDA #$00
DA35  85 26       STA $26
DA37  85 27       STA $27
DA39  85 28       STA $28       Funktionsregister löschen
DA3B  85 29       STA $29
DA3D  A5 70       LDA $70
DA3F  20 59 DA    JSR $DA59     bitweise Multiplikation
DA42  A5 65       LDA $65
DA44  20 59 DA    JSR $DA59     bitweise Multiplikation
DA47  A5 64       LDA $64
DA49  20 59 DA    JSR $DA59     bitweise Multiplikation
```

```
DA4C   A5 63       LDA $63
DA4E   20 59 DA    JSR $DA59        bitweise Multiplikation
DA51   A5 62       LDA $62
DA53   20 5E DA    JSR $DA5E        bitweise Multiplikation
DA56   4C 8F DB    JMP $DB8F        Register nach FAC

*****************************       bitweise Multiplikation
DA59   D0 03       BNE $DA5E
DA5B   4C 83 D9    JMP $D983        rechtsverschieben eines Registers
DA5E   4A          LSR
DA5F   09 80       ORA #$80
DA61   A8          TAY
DA62   90 19       BCC $DA7D
DA64   18          CLC
DA65   A5 29       LDA $29
DA67   65 6D       ADC $6D
DA69   85 29       STA $29
DA6B   A5 28       LDA $28
DA6D   65 6C       ADC $6C
DA6F   85 28       STA $28
DA71   A5 27       LDA $27
DA73   65 6B       ADC $6B
DA75   85 27       STA $27
DA77   A5 26       LDA $26
DA79   65 6A       ADC $6A
DA7B   85 26       STA $26
DA7D   66 26       ROR $26
DA7F   66 27       ROR $27
DA81   66 28       ROR $28
DA83   66 29       ROR $29
DA85   66 70       ROR $70
DA87   98          TYA
DA88   4A          LSR
DA89   D0 D6       BNE $DA61
DA8B   60          RTS

*****************************       ARG = Konstante (A/Y)
DA8C   85 22       STA $22
DA8E   84 23       STY $23
DA90   A0 04       LDY #$04
DA92   B1 22       LDA ($22),Y
DA94   85 6D       STA $6D
DA96   88          DEY
DA97   B1 22       LDA ($22),Y
DA99   85 6C       STA $6C
DA9B   88          DEY
DA9C   B1 22       LDA ($22),Y
DA9E   85 6B       STA $6B
DAA0   88          DEY
DAA1   B1 22       LDA ($22),Y
DAA3   85 6E       STA $6E
DAA5   45 66       EOR $66
DAA7   85 6F       STA $6F
DAA9   A5 6E       LDA $6E
DAAB   09 80       ORA #$80
DAAD   85 6A       STA $6A
DAAF   88          DEY
DAB0   B1 22       LDA ($22),Y
DAB2   85 69       STA $69
DAB4   A5 61       LDA $61
```

```
DAB6  60          RTS

DAB7  A5 69       LDA $69
DAB9  F0 1F       BEQ $DADA
DABB  18          CLC
DABC  65 61       ADC $61
DABE  90 04       BCC $DAC4
DAC0  30 1D       BMI $DADF
DAC2  18          CLC
DAC3  2C          .BYTE $2C
DAC4  10 14       BPL $DADA
DAC6  69 80       ADC #$80
DAC8  85 61       STA $61
DACA  D0 03       BNE $DACF
DACC  4C FB D8    JMP $D8FB      FAC = 0

DACF  A5 6F       LDA $6F
DAD1  85 66       STA $66
DAD3  60          RTS

DAD4  A5 66       LDA $66
DAD6  49 FF       EOR #$FF
DAD8  30 05       BMI $DADF
DADA  68          PLA
DADB  68          PLA
DADC  4C F7 D8    JMP $D8F7      FAC = 0

DADF  4C 7E D9    JMP $D97E      gibt 'overflow'

*****************************    FAC = FAC * 10
DAE2  20 0C DC    JSR $DC0C      FAC runden und nach ARG
DAE5  AA          TAX
DAE6  F0 10       BEQ $DAF8      FAC = 0, dann fertig
DAE8  18          CLC
DAE9  69 02       ADC #$02       Exponent plus 2 entspricht mal 4
DAEB  B0 F2       BCS $DADF      überlauf ?
DAED  A2 00       LDX #$00
DAEF  86 6F       STX $6F
DAF1  20 77 D8    JSR $D877      FAC = FAC + ARG  entspricht mal 5
DAF4  E6 61       INC $61        Exponent erhöhen, entspricht mal 10
DAF6  F0 E7       BEQ $DADF      überlauf, dann 'overflow'
DAF8  60          RTS

*****************************
DAF9  84 20 00 00 00             Fließkommakonstante 10

*****************************    FAC = FAC / 10
DAFE  20 0C DC    JSR $DC0C      FAC runden und nach ARG
DB01  A9 F9       LDA #$F9
DB03  A0 DA       LDY #$DA       Zeiger auf Konstante 10
DB05  A2 00       LDX #$00
DB07  86 6F       STX $6F
DB09  20 A2 DB    JSR $DBA2      Konstante 10 nach FAC
DB0C  4C 12 DB    JMP $DB12      FAC = ARG / FAC

*****************************    FAC = Konstante (A/Y) / FAC
DB0F  20 8C DA    JSR $DA8C      Konstante (A/Y) nach ARG

*****************************    Division  FAC = ARG / FAC
DB12  F0 76       BEQ $DB8A      FAC = 0, 'division by zero'
```

```
DB14  20 1B DC   JSR $DC1B     FAC runden
DB17  A9 00      LDA #$00
DB19  38         SEC
DB1A  E5 61      SBC $61
DB1C  85 61      STA $61
DB1E  20 B7 DA   JSR $DAB7     Exponent des Ergebnisses bestimmen
DB21  E6 61      INC $61
DB23  F0 BA      BEQ $DADF     Exponentenüberlauf, 'overflow'
DB25  A2 FC      LDX #$FC      Zeiger auf Funktionsregister
DB27  A9 01      LDA #$01
DB29  A4 6A      LDY $6A
DB2B  C4 62      CPY $62
DB2D  D0 10      BNE $DB3F
DB2F  A4 6B      LDY $6B
DB31  C4 63      CPY $63       ARG mit FAC byteweise vergleichen
DB33  D0 0A      BNE $DB3F
DB35  A4 6C      LDY $6C
DB37  C4 64      CPY $64
DB39  D0 04      BNE $DB3F
DB3B  A4 6D      LDY $6D
DB3D  C4 65      CPY $65
DB3F  08         PHP
DB40  2A         ROL
DB41  90 09      BCC $DB4C
DB43  E8         INX
DB44  95 29      STA $29,X
DB46  F0 32      BEQ $DB7A
DB48  10 34      BPL $DB7E
DB4A  A9 01      LDA #$01
DB4C  28         PLP
DB4D  B0 0E      BCS $DB5D
DB4F  06 6D      ASL $6D
DB51  26 6C      ROL $6C
DB53  26 6B      ROL $6B
DB55  26 6A      ROL $6A
DB57  B0 E6      BCS $DB3F
DB59  30 CE      BMI $DB29
DB5B  10 E2      BPL $DB3F
DB5D  A8         TAY
DB5E  A5 6D      LDA $6D
DB60  E5 65      SBC $65
DB62  85 6D      STA $6D
DB64  A5 6C      LDA $6C
DB66  E5 64      SBC $64
DB68  85 6C      STA $6C
DB6A  A5 6B      LDA $6B
DB6C  E5 63      SBC $63
DB6E  85 6B      STA $6B
DB70  A5 6A      LDA $6A
DB72  E5 62      SBC $62
DB74  85 6A      STA $6A
DB76  98         TYA
DB77  4C 4F DB   JMP $DB4F
DB7A  A9 40      LDA #$40
DB7C  D0 CE      BNE $DB4C
DB7E  0A         ASL
DB7F  0A         ASL
DB80  0A         ASL           Akku = Akku * 64
DB81  0A         ASL
DB82  0A         ASL
```

```
DB83  0A          ASL
DB84  85 70       STA $70
DB86  28          PLP
DB87  4C 8F DB    JMP $DB8F      Hilfsregister nach FAC

DB8A  A2 14       LDX #$14       Nummer für 'division by zero'
DB8C  4C 37 C4    JMP $C437      Fehlermeldung ausgeben

*****************************    Hilfsregister ($26-$29) nach FAC
DB8F  A5 26       LDA $26
DB91  85 62       STA $62
DB93  A5 27       LDA $27
DB95  85 63       STA $63
DB97  A5 28       LDA $28
DB99  85 64       STA $64
DB9B  A5 29       LDA $29
DB9D  85 65       STA $65
DB9F  4C D7 D8    JMP $D8D7      FAC linksbündig machen

*****************************    Konstante (A/Y) nach FAC übertragen
DBA2  85 22       STA $22
DBA4  84 23       STY $23        Zeiger setzen
DBA6  A0 04       LDY #$04
DBA8  B1 22       LDA ($22),Y
DBAA  85 65       STA $65
DBAC  88          DEY
DBAD  B1 22       LDA ($22),Y    Mantisse
DBAF  85 64       STA $64
DBB1  88          DEY
DBB2  B1 22       LDA ($22),Y
DBB4  85 63       STA $63
DBB6  88          DEY
DBB7  B1 22       LDA ($22),Y
DBB9  85 66       STA $66
DBBB  09 80       ORA #$80       Vorzeichen Mantisse
DBBD  85 62       STA $62
DBBF  88          DEY
DBC0  B1 22       LDA ($22),Y
DBC2  85 61       STA $61        Exponent
DBC4  84 70       STY $70
DBC6  60          RTS

*****************************    FAC nach Akku#4 übertragen
DBC7  A2 5C       LDX #$5C       Adresse low Akku#4
DBC9  2C          .BYTE $2C

*****************************    FAC nach Akku#3 übertragen
DBCA  A2 57       LDX #$57       Adresse low Akku#3
DBCC  A0 00       LDY #$00       Adresse high
DBCE  F0 04       BEQ $DBD4      unbedingter Sprung

*****************************    FAC nach Variable übertragen
DBD0  A6 49       LDX $49
DBD2  A4 4A       LDY $4A        Variablenadresse
DBD4  20 1B DC    JSR $DC1B      FAC runden
DBD7  86 22       STX $22
DBD9  84 23       STY $23        Zeiger auf Zieladresse
DBDB  A0 04       LDY #$04
DBDD  A5 65       LDA $65
DBDF  91 22       STA ($22),Y
```

```
DBE1  88        DEY
DBE2  A5 64     LDA $64       Mantisse
DBE4  91 22     STA ($22),Y
DBE6  88        DEY
DBE7  A5 63     LDA $63
DBE9  91 22     STA ($22),Y
DBEB  88        DEY
DBEC  A5 66     LDA $66
DBEE  09 7F     ORA #$7F      Vorzeichen auf Speicherformat bringen
DBF0  25 62     AND $62
DBF2  91 22     STA ($22),Y
DBF4  88        DEY
DBF5  A5 61     LDA $61       Exponent
DBF7  91 22     STA ($22),Y
DBF9  84 70     STY $70
DBFB  60        RTS

******************************  ARG nach FAC übertragen
DBFC  A5 6E     LDA $6E
DBFE  85 66     STA $66
DC00  A2 05     LDX #$05      5 Bytes
DC02  B5 68     LDA $68,X
DC04  95 60     STA $60,X
DC06  CA        DEX
DC07  D0 F9     BNE $DC02
DC09  86 70     STX $70
DC0B  60        RTS

******************************  FAC nach ARG übertragen
DC0C  20 1B DC  JSR $DC1B     FAC runden
DC0F  A2 06     LDX #$06
DC11  B5 60     LDA $60,X
DC13  95 68     STA $68,X
DC15  CA        DEX
DC16  D0 F9     BNE $DC11
DC18  86 70     STX $70
DC1A  60        RTS

******************************  FAC runden
DC1B  A5 61     LDA $61       Exponent
DC1D  F0 FB     BEQ $DC1A     null, dann fertig
DC1F  06 70     ASL $70
DC21  90 F7     BCC $DC1A     Rundungsstelle größer $7F ?
DC23  20 6F D9  JSR $D96F     Mantisse um eins erhöhen
DC26  D0 F2     BNE $DC1A     jetzt null ?
DC28  4C 38 D9  JMP $D938     nach rechts verschieben, Exponent erhöhen

******************************  Vorzeichen von FAC holen
DC2B  A5 61     LDA $61       Exponent null ?
DC2D  F0 09     BEQ $DC38     ja
DC2F  A5 66     LDA $66
DC31  2A        ROL
DC32  A9 FF     LDA #$FF      negativ
DC34  B0 02     BCS $DC38
DC36  A9 01     LDA #$01      positiv
DC38  60        RTS

******************************  BASIC-Funktion  SGN
DC39  20 2B DC  JSR $DC2B     Vorzeichen von FAC holen
DC3C  85 62     STA $62
```

```
DC3E  A9 00       LDA #$00
DC40  85 63       STA $63
DC42  A2 88       LDX #$88
DC44  A5 62       LDA $62
DC46  49 FF       EOR #$FF
DC48  2A          ROL
DC49  A9 00       LDA #$00
DC4B  85 65       STA $65
DC4D  85 64       STA $64
DC4F  86 61       STX $61
DC51  85 70       STA $70
DC53  85 66       STA $66
DC55  4C D2 D8    JMP $D8D2

*****************************   BASIC-Funktion  ABS
DC58  46 66       LSR $66       Vorzeichenbit löschen
DC5A  60          RTS

*****************************   Vergleich Konstante (A/Y) mit FAC
DC5B  85 24       STA $24
DC5D  84 25       STY $25       Zeiger auf Konstante
DC5F  A0 00       LDY #$00
DC61  B1 24       LDA ($24),Y   Exponent
DC63  C8          INY
DC64  AA          TAX
DC65  F0 C4       BEQ $DC2B     null, dann Vorzeichen von FAC holen
DC67  B1 24       LDA ($24),Y
DC69  45 66       EOR $66
DC6B  30 C2       BMI $DC2F     verschiedene Vorzeichen
DC6D  E4 61       CPX $61
DC6F  D0 21       BNE $DC92
DC71  B1 24       LDA ($24),Y   1. Byte vergleichen
DC73  09 80       ORA #$80
DC75  C5 62       CMP $62
DC77  D0 19       BNE $DC92
DC79  C8          INY
DC7A  B1 24       LDA ($24),Y   2. Byte vergleichen
DC7C  C5 63       CMP $63
DC7E  D0 12       BNE $DC92
DC80  C8          INY
DC81  B1 24       LDA ($24),Y   3. Byte vergleichen
DC83  C5 64       CMP $64
DC85  D0 0B       BNE $DC92
DC87  C8          INY
DC88  A9 7F       LDA #$7F
DC8A  C5 70       CMP $70
DC8C  B1 24       LDA ($24),Y   4. Byte vergleichen
DC8E  E5 65       SBC $65
DC90  F0 28       BEQ $DCBA
DC92  A5 66       LDA $66
DC94  90 02       BCC $DC98
DC96  49 FF       EOR #$FF      Ergebnis kleiner, dann invertieren
DC98  4C 31 DC    JMP $DC31     Flag für Ergebnis setzen

*****************************   Umwandlung Fließkomma nach Integer
DC9B  A5 61       LDA $61       Exponent
DC9D  F0 4A       BEQ $DCE9     null ?
DC9F  38          SEC
DCA0  E9 A0       SBC #$A0      Zahl größer 32768 ?
DCA2  24 66       BIT $66
```

```
DCA4  10 09      BPL $DCAF
DCA6  AA         TAX
DCA7  A9 FF      LDA #$FF
DCA9  85 68      STA $68
DCAB  20 4D D9   JSR $D94D    Mantisse von FAC invertieren
DCAE  8A         TXA
DCAF  A2 61      LDX #$61
DCB1  C9 F9      CMP #$F9
DCB3  10 06      BPL $DCBB
DCB5  20 99 D9   JSR $D999    FAC rechtsverschieben
DCB8  84 68      STY $68
DCBA  60         RTS
DCBB  A8         TAY
DCBC  A5 66      LDA $66
DCBE  29 80      AND #$80
DCC0  46 62      LSR $62
DCC2  05 62      ORA $62
DCC4  85 62      STA $62
DCC6  20 B0 D9   JSR $D9B0
DCC9  84 68      STY $68
DCCB  60         RTS

******************************   BASIC-Funktion  INT
DCCC  A5 61      LDA $61      Exponent
DCCE  C9 A0      CMP #$A0     ganze Zahl ?
DCD0  B0 20      BCS $DCF2    ja, dann fertig
DCD2  20 9B DC   JSR $DC9B    FAC nach Integer wandeln
DCD5  84 70      STY $70
DCD7  A5 66      LDA $66
DCD9  84 66      STY $66
DCDB  49 80      EOR #$80
DCDD  2A         ROL
DCDE  A9 A0      LDA #$A0
DCE0  85 61      STA $61
DCE2  A5 65      LDA $65
DCE4  85 07      STA $07
DCE6  4C D2 D8   JMP $D8D2    FAC linksbündig machen

DCE9  85 62      STA $62      Mantisse mit Nullen füllen
DCEB  85 63      STA $63
DCED  85 64      STA $64
DCEF  85 65      STA $65
DCF1  A8         TAY
DCF2  60         RTS

******************************   Umwandlung ASCII-Zahl nach Fließkomma
DCF3  A0 00      LDY #$00
DCF5  A2 0A      LDX #$0A
DCF7  94 5D      STY $5D      Bereich $5D bis $66 löschen
DCF9  CA         DEX
DCFA  10 FB      BPL $DCF7
DCFC  90 0F      BCC $DD0D
DCFE  C9 2D      CMP #$2D     '-'
DD00  D0 04      BNE $DD06
DD02  86 67      STX $67      Flag für negativ
DD04  F0 04      BEQ $DD0A
DD06  C9 2B      CMP #$2B     '+'
DD08  D0 05      BNE $DD0F
DD0A  20 73 00   JSR $0073    CHRGET  nächstes Zeichen holen
DD0D  90 5B      BCC $DD6A    Ziffer ?
```

```
DD0F  C9 2E     CMP #$2E     '.'  Dezimalpunkt ?
DD11  F0 2E     BEQ $DD41
DD13  C9 45     CMP #$45     'E'  Exponent ?
DD15  D0 30     BNE $DD47
DD17  20 73 00  JSR $0073    CHRGET  nächstes Zeichen holen
DD1A  90 17     BCC $DD33    Ziffer ?
DD1C  C9 AB     CMP #$AB     '-'  Interpreterkode
DD1E  F0 0E     BEQ $DD2E
DD20  C9 2D     CMP #$2D     '-'
DD22  F0 0A     BEQ $DD2E
DD24  C9 AA     CMP #$AA     '+'  Interpreterkode
DD26  F0 08     BEQ $DD30
DD28  C9 2B     CMP #$2B     '+'
DD2A  F0 04     BEQ $DD30
DD2C  D0 07     BNE $DD35
DD2E  66 60     ROR $60      Bit 7 setzen
DD30  20 73 00  JSR $0073    CHRGET  nächstes Zeichen holen
DD33  90 5C     BCC $DD91
DD35  24 60     BIT $60      Bit 7 gesetzt ?
DD37  10 0E     BPL $DD47    nein
DD39  A9 00     LDA #$00
DD3B  38        SEC
DD3C  E5 5E     SBC $5E
DD3E  4C 49 DD  JMP $DD49

DD41  66 5F     ROR $5F      Aufruf durch Dezimalpunkt
DD43  24 5F     BIT $5F
DD45  50 C3     BVC $DD0A
DD47  A5 5E     LDA $5E
DD49  38        SEC
DD4A  E5 5D     SBC $5D
DD4C  85 5E     STA $5E
DD4E  F0 12     BEQ $DD62
DD50  10 09     BPL $DD5B
DD52  20 FE DA  JSR $DAFE    FAC = FAC / 10
DD55  E6 5E     INC $5E
DD57  D0 F9     BNE $DD52
DD59  F0 07     BEQ $DD62
DD5B  20 E2 DA  JSR $DAE2    FAC = FAC * 10
DD5E  C6 5E     DEC $5E
DD60  D0 F9     BNE $DD5B
DD62  A5 67     LDA $67
DD64  30 01     BMI $DD67
DD66  60        RTS
DD67  4C B4 DF  JMP $DFB4    Vorzeichenwechsel  FAC = - FAC

DD6A  48        PHA
DD6B  24 5F     BIT $5F
DD6D  10 02     BPL $DD71
DD6F  E6 5D     INC $5D
DD71  20 E2 DA  JSR $DAE2    FAC = FAC * 10
DD74  68        PLA
DD75  38        SEC
DD76  E9 30     SBC #$30     ASCII - $30 = hex
DD78  20 7E DD  JSR $DD7E    addiert nächste Stelle zu FAC
DD7B  4C 0A DD  JMP $DD0A    nächstes Zeichen
DD7E  48        PHA
DD7F  20 0C DC  JSR $DC0C    FAC nach ARG
DD82  68        PLA
DD83  20 3C DC  JSR $DC3C
```

```
DD86  A5 6E        LDA $6E
DD88  45 66        EOR $66
DD8A  85 6F        STA $6F
DD8C  A6 61        LDX $61
DD8E  4C 6A D8     JMP $D86A      FAC = FAC + ARG

DD91  A5 5E        LDA $5E        Aufruf durch 'E'
DD93  C9 0A        CMP #$0A
DD95  90 09        BCC $DDA0
DD97  A9 64        LDA #$64
DD99  24 60        BIT $60
DD9B  30 11        BMI $DDAE
DD9D  4C 7E D9     JMP $D97E      gibt 'overflow'

DDA0  0A           ASL
DDA1  0A           ASL
DDA2  18           CLC
DDA3  65 5E        ADC $5E
DDA5  0A           ASL
DDA6  18           CLC
DDA7  A0 00        LDY #$00
DDA9  71 7A        ADC ($7A),Y
DDAB  38           SEC
DDAC  E9 30        SBC #$30
DDAE  85 5E        STA $5E
DDB0  4C 30 DD     JMP $DD30      nächstes Zeichen holen

******************************    Konstanten für Fließkomma nach ASCII
DDB3  9B 3E BC 1F FD              99999999.9
DDB8  9E 6E 6B 27 FD              999999999
DDBD  9E 6E 6B 28 00              1E9

******************************    Ausgabe der Zeilennummer bei Fehlermeldung
DDC2  A9 71        LDA #$71
DDC4  A0 C3        LDY #$C3       Zeiger auf 'IN'
DDC6  20 DA DD     JSR $DDDA      String ausgeben
DDC9  A5 3A        LDA $3A
DDCB  A6 39        LDX $39        laufende Zeilennummer holen

******************************    Integerzahl (A/X) ohne Vorzeichen ausgeben
DDCD  85 62        STA $62
DDCF  86 63        STX $63        Zahl merken
DDD1  A2 90        LDX #$90
DDD3  38           SEC
DDD4  20 49 DC     JSR $DC49      Integerzahl nach Fließkomma wandeln
DDD7  20 DF DD     JSR $DDDF      FAC nach ASCII wandeln
DDDA  4C 1E CB     JMP $CB1E      ASCII-String ausgeben

******************************    FAC nach ASCII-Format und nach $100
DDDD  A0 01        LDY #$01
DDDF  A9 20        LDA #$20       ' ' Leerzeichen für positive Zahl
DDE1  24 66        BIT $66        Vorzeichen testen
DDE3  10 02        BPL $DDE7      positiv ?
DDE5  A9 2D        LDA #$2D       '-' Minus für negative Zahl
DDE7  99 FF 00     STA $00FF,Y    in Puffer schreiben
DDEA  85 66        STA $66
DDEC  84 71        STY $71
DDEE  C8           INY
DDEF  A9 30        LDA #$30       '0'
DDF1  A6 61        LDX $61        Exponent
```

```
DDF3  D0 03     BNE $DDF8
DDF5  4C 04 DF  JMP $DF04     Zahl null, dann fertig
DDF8  A9 00     LDA #$00
DDFA  E0 80     CPX #$80      FAC mit eins vergleichen
DDFC  F0 02     BEQ $DE00
DDFE  B0 09     BCS $DE09     FAC größer eins
DE00  A9 BD     LDA #$BD
DE02  A0 DD     LDY #$DD      Zeiger auf Konstante 1E9
DE04  20 28 DA  JSR $DA28     mit FAC multiplizieren
DE07  A9 F7     LDA #$F7
DE09  85 5D     STA $5D
DE0B  A9 B8     LDA #$B8
DE0D  A0 DD     LDY #$DD      Zeiger auf Konstante 999999999
DE0F  20 5B DC  JSR $DC5B     mit FAC vergleichen
DE12  F0 1E     BEQ $DE32     gleich ?
DE14  10 12     BPL $DE28     größer ?
DE16  A9 B3     LDA #$B3
DE18  A0 DD     LDY #$DD      Zeiger auf Konstante 99999999.9
DE1A  20 5B DC  JSR $DC5B     mit FAC vergleichen
DE1D  F0 02     BEQ $DE21     gleich ?
DE1F  10 0E     BPL $DE2F     größer ?
DE21  20 E2 DA  JSR $DAE2     FAC = FAC * 10
DE24  C6 5D     DEC $5D
DE26  D0 EE     BNE $DE16
DE28  20 FE DA  JSR $DAFE     FAC = FAC / 10
DE2B  E6 5D     INC $5D
DE2D  D0 DC     BNE $DE0B
DE2F  20 49 D8  JSR $D849     FAC = FAC + 0.5, runden
DE32  20 9B DC  JSR $DC9B     FAC nach Integer
DE35  A2 01     LDX #$01
DE37  A5 5D     LDA $5D
DE39  18        CLC
DE3A  69 0A     ADC #$0A
DE3C  30 09     BMI $DE47     Betrag kleiner 0.1 ?
DE3E  C9 0B     CMP #$0B
DE40  B0 06     BCS $DE48     Betrag größer 1E9 ?
DE42  69 FF     ADC #$FF
DE44  AA        TAX
DE45  A9 02     LDA #$02
DE47  38        SEC
DE48  E9 02     SBC #$02
DE4A  85 5E     STA $5E
DE4C  86 5D     STX $5D
DE4E  8A        TXA
DE4F  F0 02     BEQ $DE53
DE51  10 13     BPL $DE66
DE53  A4 71     LDY $71
DE55  A9 2E     LDA #$2E      '.'
DE57  C8        INY
DE58  99 FF 00  STA $00FF,Y
DE5B  8A        TXA
DE5C  F0 06     BEQ $DE64
DE5E  A9 30     LDA #$30      '0'
DE60  C8        INY
DE61  99 FF 00  STA $00FF,Y
DE64  84 71     STY $71
DE66  A0 00     LDY #$00      Berechnung der einzelnen Ziffern
DE68  A2 80     LDX #$80
DE6A  A5 65     LDA $65
DE6C  18        CLC
```

```
DE6D  79 19 DF   ADC $DF19,Y
DE70  85 65      STA $65
DE72  A5 64      LDA $64
DE74  79 18 DF   ADC $DF18,Y
DE77  85 64      STA $64
DE79  A5 63      LDA $63
DE7B  79 17 DF   ADC $DF17,Y
DE7E  85 63      STA $63
DE80  A5 62      LDA $62
DE82  79 16 DF   ADC $DF16,Y
DE85  85 62      STA $62
DE87  E8         INX
DE88  B0 04      BCS $DE8E
DE8A  10 DE      BPL $DE6A
DE8C  30 02      BMI $DE90
DE8E  30 DA      BMI $DE6A
DE90  8A         TXA
DE91  90 04      BCC $DE97
DE93  49 FF      EOR #$FF
DE95  69 0A      ADC #$0A
DE97  69 2F      ADC #$2F
DE99  C8         INY
DE9A  C8         INY
DE9B  C8         INY
DE9C  C8         INY
DE9D  84 47      STY $47
DE9F  A4 71      LDY $71
DEA1  C8         INY
DEA2  AA         TAX
DEA3  29 7F      AND #$7F
DEA5  99 FF 00   STA $00FF,Y
DEA8  C6 5D      DEC $5D
DEAA  D0 06      BNE $DEB2
DEAC  A9 2E      LDA #$2E      '.'
DEAE  C8         INY
DEAF  99 FF 00   STA $00FF,Y
DEB2  84 71      STY $71
DEB4  A4 47      LDY $47
DEB6  8A         TXA
DEB7  49 FF      EOR #$FF
DEB9  29 80      AND #$80
DEBB  AA         TAX
DEBC  C0 24      CPY #$24      Tabellenende bei FAC-Umwandlung
DEBE  F0 04      BEQ $DEC4
DEC0  C0 3C      CPY #$3C      Tabellenende bei TI$-Berechnung
DEC2  D0 A6      BNE $DE6A
DEC4  A4 71      LDY $71
DEC6  B9 FF 00   LDA $00FF,Y
DEC9  88         DEY
DECA  C9 30      CMP #$30      '0'
DECC  F0 F8      BEQ $DEC6
DECE  C9 2E      CMP #$2E      '.'
DED0  F0 01      BEQ $DED3
DED2  C8         INY
DED3  A9 2B      LDA #$2B      '+'
DED5  A6 5E      LDX $5E
DED7  F0 2E      BEQ $DF07
DED9  10 08      BPL $DEE3
DEDB  A9 00      LDA #$00
DEDD  38         SEC
```

```
DEDE  E5 5E        SBC $5E
DEE0  AA           TAX
DEE1  A9 2D        LDA #$2D       '-'
DEE3  99 01 01     STA $0101,Y
DEE6  A9 45        LDA #$45       'E'
DEE8  99 00 01     STA $0100,Y
DEEB  8A           TXA
DEEC  A2 2F        LDX #$2F
DEEE  38           SEC
DEEF  E8           INX
DEF0  E9 0A        SBC #$0A
DEF2  B0 FB        BCS $DEEF
DEF4  69 3A        ADC #$3A
DEF6  99 03 01     STA $0103,Y
DEF9  8A           TXA
DEFA  99 02 01     STA $0102,Y
DEFD  A9 00        LDA #$00       Puffer mit 0 abschließen
DEFF  99 04 01     STA $0104,Y
DF02  F0 08        BEQ $DF0C
DF04  99 FF 00     STA $00FF,Y
DF07  A9 00        LDA #$00
DF09  99 00 01     STA $0100,Y
DF0C  A9 00        LDA #$00
DF0E  A0 01        LDY #$01       Zeiger auf Puffer $100
DF10  60           RTS

******************************
DF11  80 00 00 00 00              Konstante 0.5 für SQR

******************************    Konstanten für Fließkomma nach ASCII
                                  32-Bit Binärzahlen mit Vorzeichen
BF16  F0 0A 1F 00                 -100 000 000
BF1A  00 98 96 80                   10 000 000
BF1E  FF F0 BD C0                   -1 000 000
BF22  00 01 86 A0                      100 000
BF26  FF FF D8 F0                      -10 000
BF2A  00 00 03 E8                        1 000
BF2E  FF FF FF 9C                        - 100
BF32  00 00 00 0A                           10
BF36  FF FF FF FF                           -1

******************************    Konstanten für Umwandlung TI nach TI$
BF3A  FF DF 0A 80                 -2 160 000
BF3E  00 03 4B C0                    216 000
BF42  FF FF 73 60                    -36 000
BF46  00 00 0E 10                      3 600
BF4A  FF FF FB A8                      - 600
BF4E  00 00 00 3C                         60
BF52  EC

BF53  AA ....
BF70  .... AA

******************************    BASIC-Funktion  SQR
DF71  20 0C DC     JSR $DC0C      FAC runden und nach ARG
DF74  A9 11        LDA #$11
DF76  A0 DF        LDY #$DF       Zeiger auf Konstante 0.5

******************************    Potenzierung  FAC = ARG hoch Konstante (A/Y)
DF78  20 A2 DB     JSR $DBA2      Konstante (A/Y) nach ARG
```

```
*****************************   Potenzierung  FAC = ARG hoch FAC
DF7B  F0 70       BEQ $DFED
DF7D  A5 69       LDA $69       Exponent ARG gleich Basis
DF7F  D0 03       BNE $DF84
DF81  4C F9 D8    JMP $D8F9     null, dann fertig
DF84  A2 4E       LDX #$4E
DF86  A0 00       LDY #$00      Zeiger auf Hilfsakku
DF88  20 D4 DB    JSR $DBD4     FAC nach Hilfsakku
DF8B  A5 6E       LDA $6E       Exponent FAC gleich Potenzexponent
DF8D  10 0F       BPL $DF9E     kleiner als eins ?
DF8F  20 CC DC    JSR $DCCC     INT-Funktion
DF92  A9 4E       LDA #$4E
DF94  A0 00       LDY #$00      Zeiger auf Hilfsakku
DF96  20 5B DC    JSR $DC5B     mit FAC vergleichen
DF99  D0 03       BNE $DF9E
DF9B  98          TYA
DF9C  A4 07       LDY $07
DF9E  20 FE DB    JSR $DBFE     ARG nach FAC
DFA1  98          TYA
DFA2  48          PHA
DFA3  20 EA D9    JSR $D9EA     LOG-Funktion
DFA6  A9 4E       LDA #$4E
DFA8  A0 00       LDY #$00      Zeiger auf Hilfsakku
DFAA  20 28 DA    JSR $DA28     mit FAC multiplizieren
DFAD  20 ED DF    JSR $DFED     EXP-Funktion
DFB0  68          PLA
DFB1  4A          LSR
DFB2  90 0A       BCC $DFBE
DFB4  A5 61       LDA $61        Vorzeichenwechsel  FAC = - FAC
DFB6  F0 06       BEQ $DFBE
DFB8  A5 66       LDA $66
DFBA  49 FF       EOR #$FF
DFBC  85 66       STA $66
DFBE  60          RTS

*****************************   Konstanten für EXP
DFBF  81 38 AA 3B 29            1.44269504  = 1/LOG(2)
DFC4  07                        7 = Polynomgrad, 8 Koeffizienten
DFC5  71 34 58 3E 56            2.14987637E-5
DFCA  74 16 7E B3 1B            1.4352314E-4
DFCF  77 2F EE E3 85            1.34226348E-3
DFD4  7A 1D 84 1C 2A            9.614011701E-3
DFD9  7C 63 59 58 0A            .0555051269
DFDE  7E 75 FD E7 C6            .240226385
DFE3  80 31 72 18 10            .693147186
DFE8  81 00 00 00 00            1

*****************************   BASIC-Funktion  EXP
DFED  A9 BF       LDA #$BF
DFEF  A0 DF       LDY #$DF      Zeiger auf Konstante 1/LOG(2)
DFF1  20 28 DA    JSR $DA28     mit FAC multiplizieren
DFF4  A5 70       LDA $70
DFF6  69 50       ADC #$50
DFF8  90 03       BCC $DFFD
DFFA  20 23 DC    JSR $DC23     Mantisse von FAC um eins erhöhen
DFFD  85 56       STA $56
DFFF  20 0F DC    JSR $DC0F     FAC nach ARG bringen
E002  A5 61       LDA $61       Exponent
E004  C9 88       CMP #$88      Zahl größer 128 ?
E006  90 03       BCC $E00B
```

```
E008  20 D4 DA    JSR $DAD4     falls positiv 'overflow'
E00B  20 CC DC    JSR $DCCC     INT-Funktion
E00E  A5 07       LDA $07
E010  18          CLC
E011  69 81       ADC #$81
E013  F0 F3       BEQ $E008     gleich 127 ?
E015  38          SEC
E016  E9 01       SBC #$01
E018  48          PHA
E019  A2 05       LDX #$05
E01B  B5 69       LDA $69,X
E01D  B4 61       LDY $61       FAC und ARG vertauschen
E01F  95 61       STA $61,X
E021  94 69       STY $69
E023  CA          DEX
E024  10 F5       BPL $E01B
E026  A5 56       LDA $56
E028  85 70       STA $70
E02A  20 53 D8    JSR $D853     ARG - FAC
E02D  20 B4 DF    JSR $DFB4     Vorzeichenwechsel
E030  A9 C4       LDA #$C4
E032  A0 DF       LDY #$DF      Zeiger auf Polynomkoeffizienten
E034  20 56 E0    JSR $E056     Polynom berechnen
E037  A9 00       LDA #$00
E039  85 6F       STA $6F
E03B  68          PLA
E03C  20 B9 DA    JSR $DAB9     Exponenten von FAC und ARG addieren
E03F  60          RTS

****************************    Polynomberechnung  y=a1*x+a2*x^3+a3*x^5+...
E040  85 71       STA $71
E042  84 72       STY $72       Zeiger auf Polynomkoeffizienten
E044  20 CA DB    JSR $DBCA     FAC nach Akku#3 bringen
E047  A9 57       LDA #$57      Zeiger auf Akku#3
E049  20 28 DA    JSR $DA28     FAC * Akku#3 (quadrieren)
E04C  20 5A E0    JSR $E05A     Polynomberechnung
E04F  A9 57       LDA #$57
E051  A0 00       LDY #$00      Zeiger auf Akku#3
E053  4C 28 DA    JMP $DA28     FAC = FAC * Akku#3

******************************* Polynomberechnung  y=a0+a1*x+a2*x^2+a3*x^3+...
E056  85 71       STA $71
E058  84 72       STY $72       Zeiger auf Polynomgrad
E05A  20 C7 DB    JSR $DBC7     FAC nach Akuu#4 bringen
E05D  B1 71       LDA ($71),Y   Polynomgrad
E05F  85 67       STA $67       als Zähler benutzen
E061  A4 71       LDY $71
E063  C8          INY
E064  98          TYA           Zeiger erhöhen, zeigt dann
E065  D0 02       BNE $E069     auf ersten Koeffizienten
E067  E6 72       INC $72
E069  85 71       STA $71
E06B  A4 72       LDY $72
E06D  20 28 DA    JSR $DA28     FAC = Konstante (A/Y) * FAC
E070  A5 71       LDA $71
E072  A4 72       LDY $72
E074  18          CLC
E075  69 05       ADC #$05      Zeiger um 5 erhöhen - nächste Zahl
E077  90 01       BCC $E07A
E079  C8          INY
```

```
E07A  85 71      STA $71
E07C  84 72      STY $72
E07E  20 67 D8   JSR $D867     FAC = Konstante (A/Y) + FAC
E081  A9 5C      LDA #$5C
E083  A0 00      LDY #$00      Zeiger auf Akku#4
E085  C6 67      DEC $67       Zähler erniedrigen
E087  D0 E4      BNE $E06D
E089  60         RTS

******************************  Konstanten für RND
E08A  98 35 44 7A 00            11 879 546
E08F  68 28 B1 46 00            3.927 677 74E-4

******************************  BASIC-Funktion  RND
E094  20 2B DC   JSR $DC2B     Vorzeichen von FAC holen
E097  30 37      BMI $E0D0     negativ ?
E099  D0 20      BNE $E0BB
E09B  20 F3 FF   JSR $FFF3     Basisadresse VIA holen
E09E  86 22      STX $22
E0A0  84 23      STY $23
E0A2  A0 04      LDY #$04
E0A4  B1 22      LDA ($22),Y
E0A6  85 62      STA $62
E0A8  C8         INY
E0A9  B1 22      LDA ($22),Y
E0AB  85 64      STA $64       Timer-Werte als RND-Wert holen
E0AD  A0 08      LDY #$08
E0AF  B1 22      LDA ($22),Y
E0B1  85 63      STA $63
E0B3  C8         INY
E0B4  B1 22      LDA ($22),Y
E0B6  85 65      STA $65
E0B8  4C E0 E0   JMP $E0E0
E0BB  A9 8B      LDA #$8B
E0BD  A0 00      LDY #$00      Zeiger auf letzten RND-Wert
E0BF  20 A2 DB   JSR $DBA2     nach FAC holen
E0C2  A9 8A      LDA #$8A
E0C4  A0 E0      LDY #$E0      Zeiger auf Konstante
E0C6  20 28 DA   JSR $DA28     FAC = Konstante (A/Y) * FAC
E0C9  A9 8F      LDA #$8F
E0CB  A0 E0      LDY #$E0      Zeiger auf Konstante
E0CD  20 67 D8   JSR $D867     FAC = Konstante (A/Y) + FAC
E0D0  A6 65      LDX $65
E0D2  A5 62      LDA $62
E0D4  85 65      STA $65
E0D6  86 62      STX $62       Stellen im FAC vertauschen
E0D8  A6 63      LDX $63
E0DA  A5 64      LDA $64
E0DC  85 63      STA $63
E0DE  86 64      STX $64
E0E0  A9 00      LDA #$00
E0E2  85 66      STA $66
E0E4  A5 61      LDA $61
E0E6  85 70      STA $70
E0E8  A9 80      LDA #$80      Exponent
E0EA  85 61      STA $61
E0EC  20 D7 D8   JSR $D8D7     FAC linksbündig machen
E0EF  A2 8B      LDX #$8B
E0F1  A0 00      LDY #$00      neuen RND-Wert
E0F3  4C D4 DB   JMP $DBD4     speichern
```

```
******************************   Fehlerauswertung nach I/O-Routinen
E0F6  C9 F0      CMP #$F0      RS 232 OPEN/CLOSE ?
E0F8  D0 07      BNE $E101     nein
E0FA  84 38      STY $38
E0FC  86 37      STX $37       BASIC-RAM Ende neu setzen
E0FE  4C 63 C6   JMP $C663     zum CLR-Befehl
E101  AA         TAX           Fehlernummer nach X
E102  D0 02      BNE $E106     kein Abbruch durch STOP ?
E104  A2 1E      LDX #$1E      Nummer für 'break'
E106  4C 37 C4   JMP $C437     Fehlermeldung ausgeben
*****************************  BASIC  BSOUT
E109  20 D2 FF   JSR $FFD2     ein Zeichen ausgeben
E10C  B0 E8      BCS $E0F6     Fehler ?
E10E  60         RTS

*****************************  BASIC  BASIN
E10F  20 CF FF   JSR $FFCF     ein Zeichen holen
E112  B0 E2      BCS $E0F6     Fehler ?
E114  60         RTS

*****************************  BASIC  CKOUT
E115  20 C9 FF   JSR $FFC9     Ausgabegerät setzen
E118  B0 DC      BCS $E0F6     Fehler ?
E11A  60         RTS

*****************************  BASIC  CHKIN
E11B  20 C6 FF   JSR $FFC6     Eingabegerät setzen
E11E  B0 D6      BCS $E0F6     Fehler ?
E120  60         RTS

*****************************  BASIC  GETIN
E121  20 E4 FF   JSR $FFE4     ein Zeichen holen
E124  B0 D0      BCS $E0F6     Fehler ?
E126  60         RTS

*****************************  BASIC-Befehl  SYS
E127  20 8A CD   JSR $CD8A     FRMNUM  numerischen Ausdruck holen
E12A  20 F7 D7   JSR $D7F7     in Adressformat wandeln, nach $14/$15
E12D  A9 E1      LDA #$E1
E12F  48         PHA
E130  A9 43      LDA #$43      Rücksprungadresse
E132  48         PHA
E133  AD 0F 03   LDA $030F     Status,
E136  48         PHA
E137  AD 0C 03   LDA $030C     Akku,
E13A  AE 0D 03   LDX $030D     X-Register und
E13D  AC 0E 03   LDY $030E     Y-Register übergeben
E140  28         PLP
E141  6C 14 00   JMP ($0014)   Funktion ausführen
E144  08         PHP           Status merken
E145  8D 0C 03   STA $030C     Akku,
E148  8E 0D 03   STX $030D     X-Register,
E14B  8C 0E 03   STY $030E     Y-Register und
E14E  68         PLA
E14F  8D 0F 03   STA $030F     Status wieder speichern
E152  60         RTS

*****************************  BASIC-Befehl  SAVE
E153  20 D1 E1   JSR $E1D1     Parameter holen
E156  A6 2D      LDX $2D
```

```
E158  A4 2E       LDY $2E      Endadresse gleich BASIC-Programmende
E15A  A9 2B       LDA #$2B     Startadresse gleich BASIC-Programmstart
E15C  20 D8 FF    JSR $FFD8    Save-Routine
E15F  B0 95       BCS $E0F6    Fehler ?
E161  60          RTS

******************************  BASIC-Befehl  VERIFY
E162  A9 01       LDA #$01     Verify-Flag
E163  2C          .BYTE $2C

******************************  BASIC-Befehl  LOAD
E164  A9 00       LDA #$00     Load-Flag
E167  85 0A       STA $0A      merken
E169  20 D1 E1    JSR $E1D1    Parameter holen
E16C  A5 0A       LDA $0A      Flag
E16E  A6 2B       LDX $2B
E170  A4 2C       LDY $2C      Startadresse gleich BASIC-Programmstart
E172  20 D5 FF    JSR $FFD5    Load-Routine
E175  B0 57       BCS $E1CE    Fehler ?
E177  A5 0A       LDA $0A      Load/Verify-Flag
E179  F0 1A       BEQ $E195    Load ?
E17B  A2 1C       LDX #$1C     Nummer für 'verify error'
E17D  20 B7 FF    JSR $FFB7    Status holen
E180  29 10       AND #$10     Fehlerbit isolieren
E182  F0 03       BEQ $E187    kein Fehler ?
E184  4C 37 C4    JMP $C437    Fehlermeldung ausgeben
E187  A5 7A       LDA $7A
E189  C9 02       CMP #$02     Direkt-Modus ?
E18B  F0 07       BEQ $E194    ja, dann fertig
E18D  A9 64       LDA #$64
E18F  A0 C3       LDY #$C3     Zeiger auf 'ok'
E191  4C 1E CB    JMP $CB1E    String ausgeben
E194  60          RTS
E195  20 B7 FF    JSR $FFB7    Status holen
E198  29 BF       AND #$BF     EOF-Bit löschen
E19A  F0 05       BEQ $E1A1    kein Fehler ?
E19C  A2 1D       LDX #$1D     Nummer für 'load error'
E19E  4C 37 C4    JMP $C437    Fehlermeldung ausgeben
E1A1  A5 7B       LDA $7B
E1A3  C9 02       CMP #$02     Direkt-Modus ?
E1A5  D0 0E       BNE $E1B5    nein, dann weiter
E1A7  86 2D       STX $2D
E1A9  84 2E       STY $2E      Endadresse gleich Programmende
E1AB  A9 76       LDA #$76
E1AD  A0 C3       LDY #$C3     Zeiger auf 'ready.'
E1AF  20 1E CB    JSR $CB1E    String ausgeben
E1B2  4C 2A C5    JMP $C52A    Programmzeilen neu binden, CLR
E1B5  20 8E C6    JSR $C68E    CHRGET-Pointer auf Programmstart
E1B8  4C 76 E4    JMP $E476    zum BASIC-Warmstart

******************************  BASIC-Befehl  OPEN
E1BB  20 16 E2    JSR $E216    Parameter holen
E1BE  20 C0 FF    JSR $FFC0    Open-Routine
E1C1  B0 0B       BCS $E1CE    Fehler ?
E1C3  60          RTS

******************************  BASIC-Befehl  CLOSE
E1C4  20 16 E2    JSR $E216    Parameter holen
E1C7  A5 49       LDA $49      Filenummer
E1C9  20 C3 FF    JSR $FFC3    Close-Routine
```

```
E1CC  90 C6       BCC $E194     ok
E1CE  4C F6 E0    JMP $E0F6     zur Fehlerauswertung

*****************************   Parameter für LOAD und SAVE holen
E1D1  A9 00       LDA #$00      Default für Länge des Filenamens
E1D3  20 BD FF    JSR $FFBD     Filenamenparameter setzen
E1D6  A2 01       LDX #$01      Default für Gerätenummer
E1D8  A0 00       LDY #$00         "        Sekundäradresse
E1DA  20 BA FF    JSR $FFBA     Fileparameter setzen
E1DD  20 03 E2    JSR $E203     weitere Zeichen ?
E1E0  20 54 E2    JSR $E254     Filenamen holen
E1E3  20 03 E2    JSR $E203     weitere Zeichen ?
E1E6  20 FD E1    JSR $E1FD     Primäradresse holen
E1E9  A0 00       LDY #$00
E1EB  86 49       STX $49
E1ED  20 BA FF    JSR $FFBA     Fileparameter setzen
E1F0  20 03 E2    JSR $E203     weitere Zeichen ?
E1F3  20 FD E1    JSR $E1FD     Sekundäradresse holen
E1F6  8A          TXA
E1F7  A8          TAY
E1F8  A6 49       LDX $49
E1FA  4C BA FF    JMP $FFBA     Fileparameter setzen

*****************************
E1FD  20 0B E2    JSR $E20B     prüft auf Komma und weitere Zeichen
E200  4C 9E D7    JMP $D79E     holt Byte-Wert nach X

*****************************   prüft auf weitere Zeichen
E203  20 79 00    JSR $0079     CHRGOT  laufendes Zeichen holen
E206  D0 02       BNE $E20A     weiteres Zeichen, dann Rückkehr
E208  68          PLA           sonst Rückkehr zur übergeordneten Routine
E209  68          PLA
E20A  60          RTS

*****************************
E20B  20 FD CE    JSR $CEFD     prüft auf Komma
E20E  20 79 00    JSR $0079     CHRGOT  laufendes Zeichen holen
E211  D0 F7       BNE $E20A     weiteres Zeichen, dann Rückkehr
E213  4C 08 CF    JMP $CF08     sonst 'syntax error'

*****************************   Parameter für OPEN und CLOSE holen
E216  A9 00       LDA #$00      Default für Länge des Filenames
E218  20 BD FF    JSR $FFBD     Filenamenparameter setzen
E21B  20 0E E2    JSR $E20E     weitere Zeichen ?
E21E  20 9E D7    JSR $D79E     holt logische Filenummer
E221  86 49       STX $49
E223  8A          TXA
E224  A2 01       LDX #$01      Default für Geräteadresse
E226  A0 00       LDY #$00         "        Sekundäradresse
E228  20 BA FF    JSR $FFBA     Fileparameter setzen
E22B  20 03 E2    JSR $E203     weitere Zeichen ?
E22E  20 FD E1    JSR $E1FD     holt Geräteadresse
E231  86 4A       STX $4A
E233  A0 00       LDY #$00
E235  A5 49       LDA $49
E237  E0 03       CPX #$03
E239  90 01       BCC $E23C
E23B  88          DEY
E23C  20 BA FF    JSR $FFBA     Fileparameter setzen
E23F  20 03 E2    JSR $E203     weiter Zeichen ?
```

```
E242  20 FD E1    JSR $E1FD    holt Sekundäradresse
E245  8A          TXA
E246  A8          TAY
E247  A6 4A       LDX $4A
E249  A5 49       LDA $49
E24B  20 BA FF    JSR $FFBA    Fileparameter setzen
E24E  20 03 E2    JSR $E203    weitere Zeichen ?
E251  20 0B E2    JSR $E20B    Komma, weitere Zeichen ?
E254  20 9E CD    JSR $CD9E    holt Filenamen
E257  20 A3 D6    JSR $D6A3    holt Stringparameter
E25A  A6 22       LDX $22
E25C  A4 23       LDY $23
E25E  4C BD FF    JMP $FFBD    setzt Filenamenparameter

*****************************  BASIC-Funktion  COS
E261  A9 DD       LDA #$DD
E263  A0 E2       LDY #$E2     Zeiger auf Konstante  Pi/2
E265  20 67 D8    JSR $D867    zu FAC addieren

*****************************  BASIC-Funktion  SIN
E268  20 0C DC    JSR $DC0C    FAC runden und nach ARG
E26B  A9 E2       LDA #$E2
E26D  A0 E2       LDY #$E2     Zeiger auf Konstante Pi*2
E26F  A6 6E       LDX $6E
E271  20 07 DB    JSR $DB07    FAC durch 2*Pi didvidieren
E274  20 0C DC    JSR $DC0C    FAC runden und nach ARG
E277  20 CC DC    JSR $DCCC    INT-Funktion
E27A  A9 00       LDA #$00
E27C  85 6F       STA $6F
E27E  20 53 D8    JSR $D853    ARG minus FAC
E281  A9 E7       LDA #$E7
E283  A0 E2       LDY #$E2     Zeiger auf Konstante 0.25
E285  20 50 D8    JSR $D850    0.25 minus FAC
E288  A5 66       LDA $66
E28A  48          PHA          Vorzeichen auf Stack
E28B  10 0D       BPL $E29A    positiv ?
E28D  20 49 D8    JSR $D849    FAC + 0.5
E290  A5 66       LDA $66      Vorzeichen
E292  30 09       BMI $E29D    negativ ?
E294  A5 12       LDA $12
E296  49 FF       EOR #$FF     Flag umdrehen
E298  85 12       STA $12
E29A  20 B4 DF    JSR $DFB4    Vorzeichen wechseln
E29D  A9 E7       LDA #$E7
E29F  A0 E2       LDY #$E2     Zeiger auf Konstante 0.25
E2A1  20 67 D8    JSR $D867    FAC + 0.25
E2A4  68          PLA          Vorzeichen holen
E2A5  10 03       BPL $E2AA    positiv ?
E2A7  20 B4 DF    JSR $DFB4    Vorzeichen wechseln
E2AA  A9 EC       LDA #$EC
E2AC  A0 E2       LDY #$E2     Zeiger auf Polynomkoeffizienten
E2AE  4C 40 E0    JMP $E040    Polynom berechnen

*****************************  BASIC-Funktion  TAN
E2B1  20 CA DB    JSR $DBCA    FAC nach Akku#3
E2B4  A9 00       LDA #$00
E2B6  85 12       STA $12      Flag setzen
E2B8  20 68 E2    JSR $E268    SIN berechnen
E2BB  A2 4E       LDX #$4E
E2BD  A0 00       LDY #$00     Zeiger auf Hilfsakku
```

```
E2BF   20 F3 E0    JSR $E0F3    FAC nach Hilfsakku
E2C2   A9 57       LDA #$57
E2C4   A0 00       LDY #$00     Zeiger auf Akku#3
E2C6   20 A2 DB    JSR $DBA2    Akku#3 nach FAC
E2C9   A9 00       LDA #$00
E2CB   85 66       STA $66      Vorzeichen
E2CD   A5 12       LDA $12      Flag
E2CF   20 D9 E2    JSR $E2D9    COS berechnen
E2D2   A9 4E       LDA #$4E
E2D4   A0 00       LDY #$00     Zeiger auf Hilfsakku (SIN)
E2D6   4C 0F DB    JMP $DB0F    durch FAC dividieren
E2D9   48          PHA
E2DA   4C 9A E2    JMP $E29A    COS berechnen

******************************  Konstanten für SIN und COS
E2DD   81 49 0F DA A2           1.57079633   Pi/2
E2E2   83 49 0F DA A2           6.28318531   2*Pi
E2E7   7F 00 00 00 00           .25
E2EC   05                       5 = Polynomgrad, 6 Koeffizienten
E2ED   84 E6 1A 2D 1B           -14.3813907
E3F2   86 28 07 FB F8            42.0077971
E3F7   87 99 68 89 01           -76.7041703
E3FC   87 23 35 DF E1            81.6052237
E301   86 A5 5D E7 28           -41.3147021
E306   83 49 0F DA A2            6.28318531   2*Pi

******************************  BASIC-Funktion  ATN
E30B   A5 66       LDA $66      Vorzeichen
E30D   48          PHA          retten
E30E   10 03       BPL $E313    positiv ?
E310   20 B4 DF    JSR $DFB4    Vorzeichen wechseln
E313   A5 61       LDA $61      Exponent
E315   48          PHA          retten
E316   C9 81       CMP #$81     Zahl mit 1 vergleichen
E318   90 07       BCC $E321    kleiner ?
E31A   A9 BC       LDA #$BC
E31C   A0 D9       LDY #$D9     Zeiger auf Konstante 1
E31E   20 0F DB    JSR $DB0F    1 durch FAC dividieren (Kehrwert)
E321   A9 3B       LDA #$3B
E323   A0 E3       LDY #$E3     Zeiger auf Polynomkoeffizienten
E325   20 40 E0    JSR $E040    Polynom berechnen
E328   68          PLA          Exponent zurückholen
E329   C9 81       CMP #$81
E32B   90 07       BCC $E334    war Zahl kleiner 1
E32D   A9 DD       LDA #$DD
E32F   A0 E2       LDY #$E2     Zeiger auf Konstante Pi/2
E331   20 50 D8    JSR $D850    Pi/2 minus FAC
E334   68          PLA          Vorzeichen holen
E335   10 03       BPL $E33A    positiv ?
E337   4C B4 DF    JMP $DFB4    Vorzeichen wechseln
E33A   60          RTS

******************************  Konstanten für ATN
E33B   0B                       11 = Polynomgrad, 12 Koeffizienten
E33C   76 B3 83 BD D3           -6.84793912E-4
E341   79 1E F4 A6 F5            4.85094216E-3
E346   7B 83 FC B0 10           -.0161117015
E34B   7C 0C 1F 67 CA            .034209638
E350   7C DE 53 CB C1           -.054279133
E355   7D 14 64 70 4C            .0724571965
```

```
E35A  7D B7 EA 51 7A           -.0898019185
E35F  7D 63 30 88 7E            .110932413
E364  7E 92 44 99 3A           -.142839808
E369  7E 4C CC 91 C7            .19999912
E36E  7F AA AA AA 13           -.333333316
E373  81 00 00 00 00            1

*****************************  BASIC-Kaltstart
E378  20 5B E4    JSR $E45B    setzt BASIC-Vektoren
E37B  20 A4 E3    JSR $E3A4    Zeropage initialisieren
E37E  20 04 E4    JSR $E404    schreibt überschrift
E381  A2 FB       LDX #$FB
E383  9A          TXS          Stackpointer initialisieren
E384  4C 74 C4    JMP $C474    zum READY-Modus

*****************************  Kopie der CHRGET-Routine
E387  E6 7A       INC $7A
E389  D0 02       BNE $E38D
E38B  E6 7B       INC $7B
E38D  AD 60 EA    LDA $EA60
E390  C9 3A       CMP #$3A
E392  B0 0A       BCS $E39E
E394  C9 20       CMP #$20
E396  F0 EF       BEQ $E387
E398  38          SEC
E399  E9 30       SBC #$30
E39B  38          SEC
E39C  E9 D0       SBC #$D0
E39E  60          RTS

*****************************  Anfangswert für RND
E39F  80 4F C7 52 58           .811635157

*****************************  RAM für BASIC initialisieren
E3A4  A9 4C       LDA #$4C     JMP
E3A6  85 54       STA $54      für Funktionen
E3A8  85 00       STA $00      für USR
E3AA  A9 48       LDA #$48
E3AC  A0 D2       LDY #$D2
E3AE  85 01       STA $01      USR-Vektor auf 'ILLEGAL QUANTITY'
E3B0  84 02       STY $02
E3B2  A9 91       LDA #$91
E3B4  A0 D3       LDY #$D3     $D391
E3B6  85 05       STA $05      als Vektor für Fest/Fließkommaumwandlung
E3B8  84 06       STY $06
E3BA  A9 AA       LDA #$AA
E3BC  A0 D1       LDY #$D1     $D1AA
E3BE  85 03       STA $03      als Vektor für Fließ/Festkommaumwandlung
E3C0  84 04       STY $04
E3C2  A2 1C       LDX #$1C
E3C4  BD 87 E3    LDA $E387,X
E3C7  95 73       STA $73,X    CHRGET-Routine ins RAM kopieren
E3C9  CA          DEX
E3CA  10 F8       BPL $E3C4
E3CC  A9 03       LDA #$03
E3CE  85 53       STA $53
E3D0  A9 00       LDA #$00
E3D2  85 68       STA $68
E3D4  85 13       STA $13      Ein/Ausgabe auf Default
E3D6  85 18       STA $18
```

```
E3D8  A2 01        LDX #$01
E3DA  8E FD 01     STX $01FD
E3DD  8E FC 01     STX $01FC
E3E0  A2 19        LDX #$19
E3E2  86 16        STX $16         Zeiger für Stringverwaltung
E3E4  38           SEC
E3E5  20 9C FF     JSR $FF9C       holt RAM Anfang
E3E8  86 2B        STX $2B
E3EA  84 2C        STY $2C         als BASIC-Start speichern
E3EC  38           SEC
E3ED  20 99 FF     JSR $FF99       holt RAM Ende
E3F0  86 37        STX $37
E3F2  84 38        STY $38         als BASIC-Ende speichern
E3F4  86 33        STX $33
E3F6  84 34        STY $34
E3F8  A0 00        LDY #$00
E3FA  98           TYA
E3FB  91 2B        STA ($2B),Y     Null an BASIC-Start
E3FD  E6 2B        INC $2B
E3FF  D0 02        BNE $E403       BASIC-Start plus eins
E401  E6 2C        INC $2C
E403  60           RTS

******************************
E404  A5 2B        LDA $2B
E406  A4 2C        LDY $2C         Zeiger auf BASIC-Start
E408  20 08 C4     JSR $C408       prüft auf Platz im Speicher
E40B  A9 36        LDA #$36
E40D  A0 E4        LDY #$E4        Zeiger auf 'cbm basic'
E40F  20 1E CB     JSR $CB1E       String ausgeben
E412  A5 37        LDA $37
E414  38           SEC             BASIC-Ende
E415  E5 2B        SBC $2B         minus
E417  AA           TAX
E418  A5 38        LDA $38         BASIC-Start gleich freier Speicherplatz
E41A  E5 2C        SBC $2C
E41C  20 CD DD     JSR $DDCD       Zahl ausgeben
E41F  A9 29        LDA #$29
E421  A0 E4        LDY #$E4        Zeiger auf 'bytes free'
E423  20 1E CB     JSR $CB1E       String ausgeben
E426  4C 44 C6     JMP $C644       zum NEW-Befehl

******************************     Systemmeldungen
E429  20 42 59 54 45 53 20          bytes
E430  46 52 45 45 0D 00 93 2A      free   *
E438  2A 2A 2A 20 43 42 4D 20      *** cbm
E440  42 41 53 49 43 20 56 32      basic v2
E448  20 2A 2A 2A 2A 0D 00          ****

******************************     Tabelle der BASIC-Vektoren
E44F  3A C4 83 C4 7C C5 1A
E457  E4 C7 86 CE

******************************     BASIC-Vektoren setzen
E45B  A2 0B        LDX #$0B
E45D  BD 4F E4     LDA $E44F,X
E460  9D 00 03     STA $0300,X
E463  CA           DEX
E464  10 F7        BPL $E45D
E466  60           RTS
```

```
*****************************   BASIC NMI-Einsprung
E467  20 CC FF    JSR $FFCC     CLRCH  I/O-Kanäle rücksetzen
E46A  A9 00       LDA #$00
E46C  85 13       STA $13       BASIC I/O Flag
E46E  20 7A C6    JSR $C67A     Stackpointer initialisieren
E471  58          CLI
E472  4C 74 C4    JMP $C474     zum Ready-Modus

*****************************
E475  E8          INX
E476  20 33 C5    JSR $C533     BASIC-Zeilen neu binden
E479  4C 77 C6    JMP $C677     RESTORE und initialisieren

E47C  FF ....
E49F  .... FF

*****************************   DAV Hi ausgeben
E4A0  AD 2C 91    LDA $912C
E4A3  29 DF       AND #$DF      Bit 5 löschen
E4A5  8D 2C 91    STA $912C
E4A8  60          RTS

*****************************   DAV Lo ausgeben
E4A9  AD 2C 91    LDA $912C
E4AC  09 20       ORA #$20      Bit 5 setzen
E4AE  8D 2C 91    STA $912C
E4B1  60          RTS

*****************************   NRFD und NDAC abfragen
E4B2  AD 1F 91    LDA $911F
E4B5  CD 1F 91    CMP $911F
E4B8  D0 F8       BNE $E4B2
E4BA  4A          LSR
E4BB  60          RTS

*****************************   'searching for name' ausgeben
E4BC  A6 B9       LDX $B9       Offset für 'searching for'
E4BE  4C 47 F6    JMP $F647     Text und Filenamen ausgeben

*****************************   Endadresse für Load setzen
E4C1  8A          TXA           Sekundäradresse
E4C2  D0 08       BNE $E4CC     ungleich null ?
E4C4  A5 C3       LDA $C3
E4C6  85 AE       STA $AE       Startadresse vom File übernehmen
E4C8  A5 C4       LDA $C4
E4CA  85 AF       STA $AF
E4CC  4C 6A F6    JMP $F66A     'loading' ausgeben

*****************************
E4CF  20 E3 F8    JSR $F8E3     Block auf Band schreiben
E4D2  90 03       BCC $E4D7
E4D4  68          PLA
E4D5  A9 00       LDA #$00
E4D7  4C 9E F3    JMP $F39E     Band-Save abschließen

E4DA  FF ....
E4FF  .... FF

*****************************   Basis-Adresse des VIAs holen
E500  A2 10       LDX #$10      Zeiger X/Y auf $9110
```

```
E502  A0 91       LDY #$91
E504  60          RTS

*****************************  Bildschirmformat holen
E505  A2 16       LDX #$16      22 Spalten
E507  A0 17       LDY #$17      23 Zeilen

E509  60          RTS

*****************************  Cursor setzen (C=0)/ holen (C=1)
E50A  B0 07       BCS $E513
E50C  86 D6       STX $D6
E50E  84 D3       STY $D3
E510  20 87 E5    JSR $E587     Cursorposition berechnen
E513  A6 D6       LDX $D6       Zeile
E515  A4 D3       LDY $D3       Spalte
E517  60          RTS

*****************************  vom RESET
E518  20 BB E5    JSR $E5BB     Standard I/O, Videocontroller initialisieren
E51B  AD 88 02    LDA $0288     Video-RAM Page
E51E  29 FD       AND #$FD
E520  0A          ASL
E521  0A          ASL           Lage des Video-RAMs berechnen
E522  09 80       ORA #$80
E524  8D 05 90    STA $9005
E527  AD 88 02    LDA $0288
E52A  29 02       AND #$02
E52C  F0 08       BEQ $E536
E52E  A9 80       LDA #$80
E530  0D 02 90    ORA $9002
E533  8D 02 90    STA $9002     Videocontroller CR2
E536  A9 00       LDA #$00
E538  8D 91 02    STA $0291     Shift/Commodore ermöglichen
E53B  85 CF       STA $CF       Cursor nicht in Blinkphase
E53D  A9 DC       LDA #$DC
E53F  8D 8F 02    STA $028F
E542  A9 EB       LDA #$EB
E544  8D 90 02    STA $0290     Zeiger auf Tastatur-Decodier-Tabelle $EBDC
E547  A9 0A       LDA #$0A
E549  8D 89 02    STA $0289     Tastaturpuffer auf 10 Zeichen begrenzen
E54C  8D 8C 02    STA $028C     Zähler für Repeatverzögerung
E54F  A9 06       LDA #$06
E551  8D 86 02    STA $0286     Augenblickliche Farbe
E554  A9 04       LDA #$04
E556  8D 8B 02    STA $028B     Repeatgeschwindigkeit für Tastatureingabe
E559  A9 0C       LDA #$0C
E55B  85 CD       STA $CD       Cursor Blinkzeit
E55D  85 CC       STA $CC       Cursor Blinkflag

*****************************  CLR SCREEN
E55F  AD 88 02    LDA $0288     Speicherseite für Video-RAM
E562  09 80       ORA #$80
E564  A8          TAY
E565  A9 00       LDA #$00
E567  AA          TAX
E568  94 D9       STY $D9,X
E56A  18          CLC
E56B  69 16       ADC #$16      22 addieren
E56D  90 01       BCC $E570
```

```
E56F  C8          INY
E570  E8          INX
E571  E0 18       CPX #$18      24, schon alle Zeilen ?
E573  D0 F3       BNE $E568
E575  A9 FF       LDA #$FF
E577  95 D9       STA $D9,X
E579  A2 16       LDX #$16      22
E57B  20 8D EA    JSR $EA8D     Bildschirmzeile löschen
E57E  CA          DEX
E57F  10 FA       BPL $E57B     schon alle Zeilen ?

****************************** Cursor HOME
E581  A0 00       LDY #$00
E583  84 D3       STY $D3       Cursorspalte
E585  84 D6       STY $D6       Cursorzeile

****************************** Cursorposition berechnen
E587  A6 D6       LDX $D6       Cursorzeile
E589  A5 D3       LDA $D3       Cursorspalte
E58B  B4 D9       LDY $D9
E58D  30 08       BMI $E597
E58F  18          CLC
E590  69 16       ADC #$16      22 für eine Zeile addieren
E592  85 D3       STA $D3
E594  CA          DEX
E595  10 F4       BPL $E58B
E597  B5 D9       LDA $D9,X     MSB für Zeilenanfang
E599  29 03       AND #$03
E59B  0D 88 02    ORA $0288     mit Video-RAM-Page verknüpfen
E59E  85 D2       STA $D2       gleich Adresse high
E5A0  BD FD ED    LDA $EDFD,X   LSB aaus Tabelle holen (Spalte in X)
E5A3  85 D1       STA $D1       als Adresse low speichern
E5A5  A9 15       LDA #$15      21
E5A7  E8          INX
E5A8  B4 D9       LDY $D9
E5AA  30 06       BMI $E5B2
E5AC  18          CLC
E5AD  69 16       ADC #$16      22 für eine Zeile addieren
E5AF  E8          INX
E5B0  10 F6       BPL $E5A8
E5B2  85 D5       STA $D5
E5B4  60          RTS

******************************
E5B5  20 BB E5    JSR $E5BB     Videocontroller initialisieren
E5B8  4C 81 E5    JMP $E581     Cursor Home

****************************** Videocontroller initialisieren
E5BB  A9 03       LDA #$03
E5BD  85 9A       STA $9A       Ausgabe auf Bildschirm
E5BF  A9 00       LDA #$00
E5C1  85 99       STA $99       Eingabe auf Tastatur
E5C3  A2 10       LDX #$10
E5C5  BD E3 ED    LDA $EDE3,X   Konstanten
E5C8  9D FF 8F    STA $8FFF,X   in Videocontroller schreiben
E5CB  CA          DEX
E5CC  D0 F7       BNE $E5C5
E5CE  60          RTS

****************************** Zeichen aus Tastaturpuffer holen
```

```
E5CF   AC 77 02    LDY $0277      erstes Zeichen holen
E5D2   A2 00       LDX #$00
E5D4   BD 78 02    LDA $0278,X
E5D7   9D 77 02    STA $0277,X    Puffer aufrücken
E5DA   E8          INX
E5DB   E4 C6       CPX $C6        mit Anzahl der Zeichen im Puffer vergleichen
E5DD   D0 F5       BNE $E5D4
E5DF   C6 C6       DEC $C6        Anzahl erniedrigen
E5E1   98          TYA            Zeichen in Akku holen
E5E2   58          CLI
E5E3   18          CLC
E5E4   60          RTS

*****************************     Warteschleife für Tastatureingabe
E5E5   20 42 E7    JSR $E742      Zeichen auf Bildschirm ausgeben
E5E8   A5 C6       LDA $C6        Anzahl der gedrückten Tasten
E5EA   85 CC       STA $CC        Flag für Cursor-Blinken
E5EC   8D 92 02    STA $0292
E5EF   F0 F7       BEQ $E5E8
E5F1   78          SEI
E5F2   A5 CF       LDA $CF        Cursor in Blinkphase ?
E5F4   F0 0C       BEQ $E602
E5F6   A5 CE       LDA $CE        Zeichen unter Cursor
E5F8   AE 87 02    LDX $0287      Farbe unter Cursor
E5FB   A0 00       LDY #$00
E5FD   84 CF       STY $CF
E5FF   20 A1 EA    JSR $EAA1      Zeichen in Bildschirm-RAM schreiben
E602   20 CF E5    JSR $E5CF      Zeichen aus Tastaturpuffer holen
E605   C9 83       CMP #$83       Kode für 'Shift/RUN' ?
E607   D0 10       BNE $E619
E609   A2 09       LDX #$09       9 Zeichen in Tastaturpuffer holen
E60B   78          SEI
E60C   86 C6       STX $C6        Zeichenzahl merken
E60E   BD F3 ED    LDA $EDF3,X    'load (cr) run (cr)'
E611   9D 76 02    STA $0276,X    in Tastaturpuffer holen
E614   CA          DEX
E615   D0 F7       BNE $E60E
E617   F0 CF       BEQ $E5E8
E619   C9 0D       CMP #$0D       'cr' ?
E61B   D0 C8       BNE $E5E5      nein, dann zurück zur Warteschleife
E61D   A4 D5       LDY $D5
E61F   84 D0       STY $D0        CR-Flag setzen
E621   B1 D1       LDA ($D1),Y    Zeichen vom Bildschirm holen
E623   C9 20       CMP #$20       Blanks am Zeilenende eleminieren
E625   D0 03       BNE $E62A
E627   88          DEY
E628   D0 F7       BNE $E621
E62A   C8          INY
E62B   84 C8       STY $C8        Position als Index merken
E62D   A0 00       LDY #$00
E62F   8C 92 02    STY $0292
E632   84 D3       STY $D3
E634   84 D4       STY $D4        Hochkommaflag rücksetzen
E636   A5 C9       LDA $C9
E638   30 1D       BMI $E657
E63A   A6 D6       LDX $D6
E63C   20 19 E7    JSR $E719
E63F   E4 C9       CPX $C9
E641   D0 14       BNE $E657
E643   D0 12       BNE $E657
```

```
E645  A5 CA      LDA $CA        letzte Spalte
E647  85 D3      STA $D3        in Spaltenzeiger bringen
E649  C5 C8      CMP $C8        mit Index vergleichen
E64B  90 0A      BCC $E657
E64D  B0 42      BCS $E691

****************************    Ein Zeichen vom Bildschirm holen
E64F  98         TYA
E650  48         PHA
E651  8A         TXA
E652  48         PHA
E653  A5 D0      LDA $D0        CR-Flag
E655  F0 91      BEQ $E5E8      nein, dann zur Warteschleife
E657  A4 D3      LDY $D3        Spalte
E659  B1 D1      LDA ($D1),Y    Zeichen vom Bildschirm holen
E65B  EA ....
E671  .... EA
E672  85 D7      STA $D7        und merken
E674  29 3F      AND #$3F
E676  06 D7      ASL $D7
E678  24 D7      BIT $D7
E67A  10 02      BPL $E67E
E67C  09 80      ORA #$80       Bildschirmcode nach ASCII konvertieren
E67E  90 04      BCC $E684
E680  A6 D4      LDX $D4
E682  D0 04      BNE $E688
E684  70 02      BVS $E688
E686  09 40      ORA #$40
E688  E6 D3      INC $D3        Cursor eins weiter setzen
E68A  20 B8 E6   JSR $E6B8      auf Hochkomma testen
E68D  C4 C8      CPY $C8        Cursor in letzter Spalte?
E68F  D0 17      BNE $E6A8
E691  A9 00      LDA #$00
E693  85 D0      STA $D0        CR-Flag rücksetzen
E695  A9 0D      LDA #$0D       'CR'-Code
E697  A6 99      LDX $99
E699  E0 03      CPX #$03       Eingabe vom Bildschirm ?
E69B  F0 06      BEQ $E6A3      ja
E69D  A6 9A      LDX $9A
E69F  E0 03      CPX #$03       Ausgabe auf Bildschirm ?
E6A1  F0 03      BEQ $E6A6      ja
E6A3  20 42 E7   JSR $E742      Zeichen auf Bildschirm schreiben
E6A6  A9 0D      LDA #$0D       'CR'-Code
E6A8  85 D7      STA $D7
E6AA  68         PLA
E6AB  AA         TAX
E6AC  68         PLA
E6AD  A8         TAY
E6AE  A5 D7      LDA $D7        Bildschirmkode
E6B0  C9 DE      CMP #$DE       mit Kode für Pi vergleichen
E6B2  D0 02      BNE $E6B6
E6B4  A9 FF      LDA #$FF       durch BASIC-Kode ersetzen
E6B6  18         CLC
E6B7  60         RTS

****************************    auf Hochkomma testen
E6B8  C9 22      CMP #$22       '"'
E6BA  D0 08      BNE $E6C4
E6BC  A5 D4      LDA $D4
E6BE  49 01      EOR #$01       Hochkommaflag umdrehen
```

```
E6C0  85 D4       STA $D4
E6C2  A9 22       LDA #$22       Kode wieder herstellen
E6C4  60          RTS

*****************************    druckendes Zeichen auf Bildschirm
E6C5  09 40       ORA #$40
E6C7  A6 C7       LDX $C7
E6C9  F0 02       BEQ $E6CD      RVS-Modus ?
E6CB  09 80       ORA #$80       ja, dann Bit 7 setzen
E6CD  A6 D8       LDX $D8        Insert-Zähler
E6CF  F0 02       BEQ $E6D3
E6D1  C6 D8       DEC $D8
E6D3  AE 86 02    LDX $0286      Farbcode
E6D6  20 A1 EA    JSR $EAA1      setzt Zeichen und Farbe
E6D9  20 EA E6    JSR $E6EA      Tabelle für Zeilenanfänge aktualisieren
E6DC  68          PLA
E6DD  A8          TAY
E6DE  A5 D8       LDA $D8        Insert-Zähler
E6E0  F0 02       BEQ $E6E4      gleich null ?
E6E2  46 D4       LSR $D4        Hochkommaflag rücksetzen
E6E4  68          PLA
E6E5  AA          TAX
E6E6  68          PLA
E6E7  18          CLC
E6E8  58          CLI
E6E9  60          RTS

*****************************    MSB für Zeilenanfänge neu berechnen
E6EA  20 FA E8    JSR $E8FA
E6ED  E6 D3       INC $D3
E6EF  A5 D5       LDA $D5
E6F1  C5 D3       CMP $D3
E6F3  B0 37       BCS $E72C
E6F5  C9 57       CMP #$57       88 Zeichen?  (4-fach Zeile)
E6F7  F0 2A       BEQ $E723
E6F9  AD 92 02    LDA $0292
E6FC  F0 03       BEQ $E701
E6FE  4C F0 E9    JMP $E9F0
E701  A6 D6       LDX $D6
E703  E0 17       CPX #$17       Cursor in letzter Zeile?
E705  90 07       BCC $E70E
E707  20 75 E9    JSR $E975      Scrolling
E70A  C6 D6       DEC $D6        Zeilenummer erniedrigen
E70C  A6 D6       LDX $D6
E70E  16 D9       ASL $D9,X
E710  56 D9       LSR $D9,X
E712  4C 5B ED    JMP $ED5B      zurück nach $E715
E715  69 16       ADC #$16       22 addieren (eine Zeile)
E717  85 D5       STA $D5
E719  B5 D9       LDA $D9,X
E71B  30 03       BMI $E720
E71D  CA          DEX
E71E  D0 F9       BNE $E719
E720  4C 7E EA    JMP $EA7E      Wert aus Tabelle holen
E723  C6 D6       DEC $D6
E725  20 C3 E8    JSR $E8C3
E728  A9 00       LDA #$00
E72A  85 D3       STA $D3
E72C  60          RTS
```

```
******************************   Rückschritt in vorhergehende Zeile
E72D  A6 D6      LDX $D6
E72F  D0 06      BNE $E737
E731  86 D3      STX $D3
E733  68         PLA
E734  68         PLA
E735  D0 A5      BNE $E6DC
E737  CA         DEX
E738  86 D6      STX $D6
E73A  20 87 E5   JSR $E587
E73D  A4 D5      LDY $D5
E73F  84 D3      STY $D3
E741  60         RTS

******************************   Ausgabe eines Zeichen auf Bildschirm
E742  48         PHA
E743  85 D7      STA $D7
E745  8A         TXA             A, X, Y retten, Zeichen nach $D7
E746  48         PHA
E747  98         TYA
E748  48         PHA
E749  A9 00      LDA #$00
E74B  85 D0      STA $D0         'CR'-Flag rücksetzen
E74D  A4 D3      LDY $D3         Cursorspalte
E74F  A5 D7      LDA $D7         Zeichen
E751  10 03      BPL $E756
E753  4C 00 E8   JMP $E800       Kode ist größer $7F
E756  C9 0D      CMP #$0D        'CR' ?
E758  D0 03      BNE $E75D
E75A  4C D8 E8   JMP $E8D8       ja , CR' ausgeben
E75D  C9 20      CMP #$20
E75F  90 10      BCC $E771       Kontrollzeichen ?
E761  C9 60      CMP #$60
E763  90 04      BCC $E769
E765  29 DF      AND #$DF
E767  D0 02      BNE $E76B
E769  29 3F      AND #$3F        Zeichen in Bildschirmcode wandeln
E76B  20 B8 E6   JSR $E6B8       auf Hochkomma testen
E76E  4C C7 E6   JMP $E6C7       und ausgeben

******************************   Steuerzeichen verarbeiten
E771  A6 D8      LDX $D8
E773  F0 03      BEQ $E778
E775  4C CB E6   JMP $E6CB
E778  C9 14      CMP #$14        'DEL'
E77A  D0 2E      BNE $E7AA       nein?
E77C  98         TYA             Cursor in Spalte Null?
E77D  D0 06      BNE $E785       nein?
E77F  20 2D E7   JSR $E72D       Rückschritt in vorhergehende Zeile
E782  4C 9F E7   JMP $E79F

E785  20 E8 E8   JSR $E8E8
E788  88         DEY
E789  84 D3      STY $D3
E78B  20 B2 EA   JSR $EAB2       Zeiger auf Farb-RAM berechnen
E78E  C8         INY
E78F  B1 D1      LDA ($D1),Y
E791  88         DEY
E792  91 D1      STA ($D1),Y
E794  C8         INY             Zeichen hinter Cursor nach links
```

```
E795  B1 F3      LDA ($F3),Y
E797  88         DEY
E798  91 F3      STA ($F3),Y
E79A  C8         INY
E79B  C4 D5      CPY $D5
E79D  D0 EF      BNE $E78E
E79F  A9 20      LDA #$20
E7A1  91 D1      STA ($D1),Y   letztes Zeichen durch Blank ersetzen
E7A3  AD 86 02   LDA $0286
E7A6  91 F3      STA ($F3),Y   und Farbe setzen
E7A8  10 4D      BPL $E7F7     fertig
E7AA  A6 D4      LDX $D4       Cursorzeile
E7AC  F0 03      BEQ $E7B1
E7AE  4C CB E6   JMP $E6CB     ungleich null ?
E7B1  C9 12      CMP #$12      'RVS ON'
E7B3  D0 02      BNE $E7B7
E7B5  85 C7      STA $C7       RVS-Flag setzen
E7B7  C9 13      CMP #$13      'HOME'
E7B9  D0 03      BNE $E7BE
E7BB  20 81 E5   JSR $E581     Cursor HOME
E7BE  C9 1D      CMP #$1D      'Cursor RIGHT'
E7C0  D0 17      BNE $E7D9      nein?
E7C2  C8         INY
E7C3  20 FA E8   JSR $E8FA
E7C6  84 D3      STY $D3
E7C8  88         DEY
E7C9  C4 D5      CPY $D5
E7CB  90 09      BCC $E7D6
E7CD  C6 D6      DEC $D6
E7CF  20 C3 E8   JSR $E8C3
E7D2  A0 00      LDY #$00
E7D4  84 D3      STY $D3       Cursorspalte speichern
E7D6  4C DC E6   JMP $E6DC     fertig
E7D9  C9 11      CMP #$11      'Cursor DOWN'
E7DB  D0 1D      BNE $E7FA
E7DD  18         CLC
E7DE  98         TYA
E7DF  69 16      ADC #$16      Cursorspalte + 22 = eine Zeile tiefer
E7E1  A8         TAY
E7E2  E6 D6      INC $D6       Cursorzeile erhöhen
E7E4  C5 D5      CMP $D5
E7E6  90 EC      BCC $E7D4     Cursorspalte mit Zeichenzahl vergleichen
E7E8  F0 EA      BEQ $E7D4     kleiner oder gleich?
E7EA  C6 D6      DEC $D6       Cursorzeile erniedrigen
E7EC  E9 16      SBC #$16      Cursorspalte - 22
E7EE  90 04      BCC $E7F4
E7F0  85 D3      STA $D3
E7F2  D0 F8      BNE $E7EC
E7F4  20 C3 E8   JSR $E8C3
E7F7  4C DC E6   JMP $E6DC     fertig

E7FA  20 12 E9   JSR $E912     prüft auf Farbcode
E7FD  4C 21 ED   JMP $ED21     prüft auf weitere Sonderzeichen
E800  EA ....    NOP
E814  .... EA    NOP
E815  29 7F      AND #$7F      Kode größer 127, Bit 7 wird gelöscht
E817  C9 7F      CMP #$7F      BASIC-Kode für PI ?
E819  D0 02      BNE $E81D     nein?
E81B  A9 5E      LDA #$5E      Bildschirmkode für PI  $5E
E81D  EA EA EA EA EA EA        NOP's
```

```
E823  C9 20     CMP #$20
E825  90 03     BCC $E82A
E827  4C C5 E6  JMP $E6C5     größer $A0, druckendes Zeichen
E82A  C9 0D     CMP #$0D      Shift 'CR'
E82C  D0 03     BNE $E831
E82E  4C D8 E8  JMP $E8D8     'CR' ausgeben
E831  A6 D4     LDX $D4
E833  D0 3F     BNE $E874     Hochkommamodus?
E835  C9 14     CMP #$14      'INST'  Zeichen einfügen
E837  D0 37     BNE $E870
E839  A4 D5     LDY $D5       Zeichen pro Zeile
E83B  B1 D1     LDA ($D1),Y   letztes Zeichen in Zeile
E83D  C9 20     CMP #$20      Blank ?
E83F  D0 04     BNE $E845     nein?
E841  C4 D3     CPY $D3       Cursor in letzter Spalte?
E843  D0 07     BNE $E84C     nein?
E845  C0 57     CPY #$57      Cursor in Pos. 87 (Zeile voll) ?
E847  F0 24     BEQ $E86D     ja, dann fertig
E849  20 EE E9  JSR $E9EE     einfügen einer Fortsetzungszeile
E84C  A4 D5     LDY $D5
E84E  20 B2 EA  JSR $EAB2     Zeiger auf Farb-RAM berechnen
E851  88        DEY
E852  B1 D1     LDA ($D1),Y
E854  C8        INY           die Zeichen ab Cursorposition
E855  91 D1     STA ($D1),Y
E857  88        DEY           um eine Stelle nach rechts
E858  B1 F3     LDA ($F3),Y
E85A  C8        INY
E85B  91 F3     STA ($F3),Y   $D1/$D2  Video-RAM
E85D  88        DEY           $F3/$F4  Farb-RAM
E85E  C4 D3     CPY $D3
E860  D0 EF     BNE $E851
E862  A9 20     LDA #$20      an augenblickliche Position
E864  91 D1     STA ($D1),Y   Blank einfügen
E866  AD 86 02  LDA $0286
E869  91 F3     STA ($F3),Y   Farbe setzen
E86B  E6 D8     INC $D8       fertig
E86D  4C DC E6  JMP $E6DC

E870  A6 D8     LDX $D8
E872  F0 05     BEQ $E879
E874  09 40     ORA #$40
E876  4C CB E6  JMP $E6CB

E879  C9 11     CMP #$11      'Cursor UP'
E87B  D0 16     BNE $E893
E87D  A6 D6     LDX $D6
E87F  F0 37     BEQ $E8B8
E881  C6 D6     DEC $D6
E883  A5 D3     LDA $D3
E885  38        SEC
E886  E9 16     SBC #$16      minus 22 gleich eine Zeile
E888  90 04     BCC $E88E
E88A  85 D3     STA $D3
E88C  10 2A     BPL $E8B8
E88E  20 87 E5  JSR $E587
E891  D0 25     BNE $E8B8
E893  C9 12     CMP #$12       'RVS OFF'
E895  D0 04     BNE $E89B
E897  A9 00     LDA #$00
```

```
E899  85 C7      STA $C7      RVS-Flag loeschen
E89B  C9 1D      CMP #$1D     'Cursor LEFT'
E89D  D0 12      BNE $E8B1
E89F  98         TYA
E8A0  F0 09      BEQ $E8AB
E8A2  20 E8 E8   JSR $E8E8
E8A5  88         DEY
E8A6  84 D3      STY $D3
E8A8  4C DC E6   JMP $E6DC

E8AB  20 2D E7   JSR $E72D    Rückschritt in vorhergehende Zeile
E8AE  4C DC E6   JMP $E6DC

E8B1  C9 13      CMP #$13     'CLR SCREEN'
E8B3  D0 06      BNE $E8BB
E8B5  20 5F E5   JSR $E55F    Bildschirm löschen
E8B8  4C DC E6   JMP $E6DC

E8BB  09 80      ORA #$80     achtes Bit wiederherstellen
E8BD  20 12 E9   JSR $E912    prüft auf Farbcode
E8C0  4C 30 ED   JMP $ED30    prüft auf weitere Sonderzeichen

*****************************  Zeilenvorschub
E8C3  46 C9      LSR $C9
E8C5  A6 D6      LDX $D6      aktuelle Zeile
E8C7  E8         INX
E8C8  E0 17      CPX #$17     Cursor in letzter Zeile?
E8CA  D0 03      BNE $E8CF    nein?
E8CC  20 75 E9   JSR $E975    Scrolling
E8CF  B5 D9      LDA $D9,X
E8D1  10 F4      BPL $E8C7
E8D3  86 D6      STX $D6
E8D5  4C 87 E5   JMP $E587

*****************************  'CR' ausgeben
E8D8  A2 00      LDX #$00
E8DA  86 D8      STX $D8      Insert-Zähler
E8DC  86 C7      STX $C7      RVS-Flag
E8DE  86 D4      STX $D4      Hochkommaflag
E8E0  86 D3      STX $D3      Cursorspalte
E8E2  20 C3 E8   JSR $E8C3
E8E5  4C DC E6   JMP $E6DC

*****************************  für Cursor LEFT
E8E8  A2 04      LDX #$04
E8EA  A9 00      LDA #$00
E8EC  C5 D3      CMP $D3
E8EE  F0 07      BEQ $E8F7
E8F0  18         CLC
E8F1  69 16      ADC #$16
E8F3  CA         DEX
E8F4  D0 F6      BNE $E8EC
E8F6  60         RTS
E8F7  C6 D6      DEC $D6
E8F9  60         RTS

*****************************  für Cursor RIGHT
E8FA  A2 04      LDX #$04
E8FC  A9 15      LDA #$15
E8FE  C5 D3      CMP $D3
```

```
E900  F0 07          BEQ $E909
E902  18             CLC
E903  69 16          ADC #$16
E905  CA             DEX
E906  D0 F6          BNE $E8FE
E908  60             RTS
E909  A6 D6          LDX $D6
E90B  E0 17          CPX #$17
E90D  F0 02          BEQ $E911
E90F  E6 D6          INC $D6
E911  60             RTS

******************************  prüft auf Farbkode
E912  A2 07          LDX #$07       8 Farbkodes
E914  DD 21 E9       CMP $E921,X    mit Tabelle vergleichen
E917  F0 04          BEQ $E91D
E919  CA             DEX
E91A  10 F8          BPL $E914
E91C  60             RTS
E91D  8E 86 02       STX $0286      Farbkode setzen
E920  60             RTS
******************************  Tabelle der Farbkodes
E921  90 05 9E EF A1 DF A6 E1

E92E  B1 E2
E930  B2 E3 B3 E4 B4 E5 B5 E6
E938  B6 E7 B7 E8 B8 E9 B9 FA
E940  BA FB BB FC BC EC BD FE
E948  BE 84 BF F7 C0 F8 DB F9
E950  DD EA DE 5E E0 5B E1 5D
E958  E2 40 B0 61 B1 78 DB 79
E960  DD 66 B6 77 C0 70 F0 71
E968  F1 72 F2 73 F3 74 F4 75
E970  F5 76 F6 7D FD

******************************  Scrolling - Bildschirm eine Zeile hoch
E975  A5 AC          LDA $AC
E977  48             PHA
E978  A5 AD          LDA $AD
E97A  48             PHA
E97B  A5 AE          LDA $AE
E97D  48             PHA
E97E  A5 AF          LDA $AF
E980  48             PHA
E981  A2 FF          LDX #$FF
E983  C6 D6          DEC $D6
E985  C6 C9          DEC $C9
E987  C6 F2          DEC $F2
E989  E8             INX
E98A  20 7E EA       JSR $EA7E
E98D  E0 16          CPX #$16
E98F  B0 0C          BCS $E99D
E991  BD FE ED       LDA $EDFE,X
E994  85 AC          STA $AC
E996  B5 DA          LDA $DA,X
E998  20 56 EA       JSR $EA56       Zeile nach oben schieben
E99B  30 EC          BMI $E989
E99D  20 8D EA       JSR $EA8D       Bildschirmzeile loeschen
E9A0  A2 00          LDX #$00
E9A2  B5 D9          LDA $D9,X
```

```
E9A4  29 7F      AND #$7F
E9A6  B4 DA      LDY $DA
E9A8  10 02      BPL $E9AC
E9AA  09 80      ORA #$80
E9AC  95 D9      STA $D9,X
E9AE  E8         INX
E9AF  E0 16      CPX #$16
E9B1  D0 EF      BNE $E9A2
E9B3  A5 EF      LDA $EF
E9B5  09 80      ORA #$80
E9B7  85 EF      STA $EF
E9B9  A5 D9      LDA $D9
E9BB  10 C4      BPL $E981
E9BD  E6 D6      INC $D6
E9BF  E6 F2      INC $F2
E9C1  A9 FB      LDA #$FB
E9C3  8D 20 91   STA $9120
E9C6  AD 21 91   LDA $9121
E9C9  C9 FE      CMP #$FE      CTRL-Taste gedrückt?
E9CB  08         PHP
E9CC  A9 F7      LDA #$F7
E9CE  8D 20 91   STA $9120
E9D1  28         PLP
E9D2  D0 0B      BNE $E9DF     nein, dann fertig
E9D4  A0 00      LDY #$00
E9D6  EA         NOP
E9D7  CA         DEX
E9D8  D0 FC      BNE $E9D6     Verzögerungsschleife
E9DA  88         DEY
E9DB  D0 F9      BNE $E9D6
E9DD  84 C6      STY $C6       Anzahl der gedrückten Tasten = Null
E9DF  A6 D6      LDX $D6       Zeilennummer
E9E1  68         PLA
E9E2  85 AF      STA $AF
E9E4  68         PLA
E9E5  85 AE      STA $AE
E9E7  68         PLA
E9E8  85 AD      STA $AD
E9EA  68         PLA
E9EB  85 AC      STA $AC
E9ED  60         RTS

******************************    Einfügen einer Fortsetzungszeile
E9EE  A6 D6      LDX $D6
E9F0  E8         INX
E9F1  B5 D9      LDA $D9,X
E9F3  10 FB      BPL $E9F0
E9F5  86 F2      STX $F2
E9F7  E0 16      CPX #$16
E9F9  F0 0D      BEQ $EA08
E9FB  90 0B      BCC $EA08
E9FD  20 75 E9   JSR $E975     Scrolling
EA00  A6 F2      LDX $F2
EA02  CA         DEX
EA03  C6 D6      DEC $D6
EA05  4C 0E E7   JMP $E70E

EA08  A5 AC      LDA $AC
EA0A  48         PHA
EA0B  A5 AD      LDA $AD
```

```
EA0D  48          PHA
EA0E  A5 AE       LDA $AE
EA10  48          PHA
EA11  A5 AF       LDA $AF
EA13  48          PHA
EA14  A2 17       LDX #$17
EA16  CA          DEX
EA17  20 7E EA    JSR $EA7E
EA1A  E4 F2       CPX $F2
EA1C  90 0E       BCC $EA2C
EA1E  F0 0C       BEQ $EA2C
EA20  BD FC ED    LDA $EDFC,X
EA23  85 AC       STA $AC
EA25  B5 D8       LDA $D8,X
EA27  20 56 EA    JSR $EA56       Zeile nach oben schieben
EA2A  30 EA       BMI $EA16
EA2C  20 8D EA    JSR $EA8D       Bildschirmzeile löschen
EA2F  A2 15       LDX #$15
EA31  E4 F2       CPX $F2
EA33  90 0F       BCC $EA44
EA35  B5 DA       LDA $DA,X
EA37  29 7F       AND #$7F
EA39  B4 D9       LDY $D9
EA3B  10 02       BPL $EA3F
EA3D  09 80       ORA #$80
EA3F  95 DA       STA $DA,X
EA41  CA          DEX
EA42  D0 ED       BNE $EA31
EA44  A6 F2       LDX $F2
EA46  20 0E E7    JSR $E70E
EA49  68          PLA
EA4A  85 AF       STA $AF
EA4C  68          PLA
EA4D  85 AE       STA $AE
EA4F  68          PLA
EA50  85 AD       STA $AD
EA52  68          PLA
EA53  85 AC       STA $AC
EA55  60          RTS

******************************    Bildschirmzeile um eins nach oben schieben
EA56  29 03       AND #$03
EA58  0D 88 02    ORA $0288       Page Video-RAM
EA5B  85 AD       STA $AD         A enthält MSB des Zeilenanfangs
EA5D  20 6E EA    JSR $EA6E
EA60  A0 15       LDY #$15        (LDA $DA,X  X=Zeilennummer )
EA62  B1 AC       LDA ($AC),Y
EA64  91 D1       STA ($D1),Y     Zeichen
EA66  B1 AE       LDA ($AE),Y
EA68  91 F3       STA ($F3),Y     und Farbe verschieben
EA6A  88          DEY
EA6B  10 F5       BPL $EA62
EA6D  60          RTS

******************************    Zeiger auf Bildschirmposition berechnen
EA6E  20 B2 EA    JSR $EAB2       Zeiger auf Farb-RAM berechnen
EA71  A5 AC       LDA $AC
EA73  85 AE       STA $AE
EA75  A5 AD       LDA $AD
EA77  29 03       AND #$03
```

```
EA79  09 94      ORA #$94
EA7B  85 AF      STA $AF
EA7D  60         RTS

******************************  LSB für Zeilenanfänge auf Tabelle holen
EA7E  BD FD ED   LDA $EDFD,X    X enthält Zeilennummer
EA81  85 D1      STA $D1
EA83  B5 D9      LDA $D9,X
EA85  29 03      AND #$03
EA87  0D 88 02   ORA $0288
EA8A  85 D2      STA $D2
EA8C  60         RTS

******************************  eine Bildschirmzeile löschen
EA8D  A0 15      LDY #$15
EA8F  20 7E EA   JSR $EA7E       Nummer der Zeile in X
EA92  20 B2 EA   JSR $EAB2       Zeiger auf Farb-RAM berechnen
EA95  A9 20      LDA #$20
EA97  91 D1      STA ($D1),Y
EA99  A9 01      LDA #$01
EA9B  91 F3      STA ($F3),Y
EA9D  88         DEY
EA9E  10 F5      BPL $EA95
EAA0  60         RTS
******************************  setzt Zeichen (A) und Farbe (X) auf Bildschirm
EAA1  A8         TAY
EAA2  A9 02      LDA #$02
EAA4  85 CD      STA $CD
EAA6  20 B2 EA   JSR $EAB2      Zeiger auf Farb-RAM berechnen
EAA9  98         TYA
EAAA  A4 D3      LDY $D3
EAAC  91 D1      STA ($D1),Y    Zeichen setzen
EAAE  8A         TXA
EAAF  91 F3      STA ($F3),Y    Farbe setzen
EAB1  60         RTS

******************************  Zeiger auf Farb-RAM berechnen
EAB2  A5 D1      LDA $D1        $D1/$D2  Zeiger auf Video-RAM
EAB4  85 F3      STA $F3
EAB6  A5 D2      LDA $D2        $F3/$F4 Zeiger auf Farb-RAM
EAB8  29 03      AND #$03
EABA  09 94      ORA #$94
EABC  85 F4      STA $F4
EABE  60         RTS

******************************  INTERRUPT-ROUTINE
EABF  20 EA FF   JSR $FFEA      Zeit weiter setzen
EAC2  A5 CC      LDA $CC        Blinkflag für Cursor gesetzt?
EAC4  D0 29      BNE $EAEF      ja?
EAC6  C6 CD      DEC $CD        Blinkzähler erniedrigen
EAC8  D0 25      BNE $EAEF      ungleich null ?
EACA  A9 14      LDA #$14
EACC  85 CD      STA $CD        Zähler auf Anfangswert 20 setzen
EACE  A4 D3      LDY $D3        Spaltenzeiger
EAD0  46 CF      LSR $CF        Blinkschalter  0  dann C=1
EAD2  AE 87 02   LDX $0287      Farbe unter Cursor
EAD5  B1 D1      LDA ($D1),Y    Zeichen unter Cursor
EAD7  B0 11      BCS $EAEA      war Blinkschalter ein ?
EAD9  E6 CF      INC $CF        Blinkschalter einschalten
EADB  85 CE      STA $CE        Zeichen unter Cursor merken
```

```
EADD  20 B2 EA    JSR $EAB2     Zeiger auf Farb-RAM berechnen
EAE0  B1 F3       LDA ($F3),Y   Farbcode holen
EAE2  8D 87 02    STA $0287     und merken
EAE5  AE 86 02    LDX $0286     Farbcode unter Cursor
EAE8  A5 CE       LDA $CE       Zeichen unter Cursor
EAEA  49 80       EOR #$80      RVS-Bit umdrehen
EAEC  20 AA EA    JSR $EAAA     Zeichen und Farbe setzen
EAEF  AD 1F 91    LDA $911F
EAF2  29 40       AND #$40
EAF4  F0 0B       BEQ $EB01     Recorder-Taste gedrückt ?
EAF6  A0 00       LDY #$00
EAF8  84 C0       STY $C0       Recorder-Flag setzen
EAFA  AD 1C 91    LDA $911C
EAFD  09 02       ORA #$02      Recorder-Motor aus
EAFF  D0 09       BNE $EB0A
EB01  A5 C0       LDA $C0
EB03  D0 0D       BNE $EB12
EB05  AD 1C 91    LDA $911C
EB08  29 FD       AND #$FD      Recorder-Motor ein
EB0A  2C 1E 91    BIT $911E
EB0D  70 03       BVS $EB12
EB0F  8D 1C 91    STA $911C
EB12  20 1E EB    JSR $EB1E     Tastatur-Abfrage
EB15  2C 24 91    BIT $9124
EB18  68          PLA
EB19  A8          TAY
EB1A  68          PLA
EB1B  AA          TAX
EB1C  68          PLA
EB1D  40          RTI

******************************  Tastaturabfrage
EB1E  A9 00       LDA #$00
EB20  8D 8D 02    STA $028D     Shift/CTRL-Flag rücksetzen
EB23  A0 40       LDY #$40      $40 gleich keine Taste gedrückt
EB25  84 CB       STY $CB       Kode für gedrückte Taste
EB27  8D 20 91    STA $9120
EB2A  AE 21 91    LDX $9121
EB2D  E0 FF       CPX #$FF      keine Taste gedrückt?
EB2F  F0 5E       BEQ $EB8F     dann fertig
EB31  A9 FE       LDA #$FE
EB33  8D 20 91    STA $9120
EB36  A0 00       LDY #$00
EB38  A9 5E       LDA #$5E
EB3A  85 F5       STA $F5       Zeiger $F5/$F6 auf Tabelle 1,$EC5E
EB3C  A9 EC       LDA #$EC
EB3E  85 F6       STA $F6
EB40  A2 08       LDX #$08      8 Matrixzeilen
EB42  AD 21 91    LDA $9121     Tastatur-Decoder-Ausgang
EB45  CD 21 91    CMP $9121     entprellen
EB48  D0 F6       BNE $EB40
EB4A  4A          LSR           Bits nacheinander ins Carry schieben
EB4B  B0 16       BCS $EB63     '1' gleich nicht gedrückt
EB4D  48          PHA
EB4E  B1 F5       LDA ($F5),Y   ASCII-Code aus Tabelle holen
EB50  C9 05       CMP #$05
EB52  B0 0C       BCS $EB60     größer gleich 5
EB54  C9 03       CMP #$03
EB56  F0 08       BEQ $EB60     'STOP'-Kode ?
EB58  0D 8D 02    ORA $028D
```

```
EB5B  8D 8D 02   STA $028D     Shift/CTRL-Flag
EB5E  10 02      BPL $EB62
EB60  84 CB      STY $CB       Nummer der Taste merken
EB62  68         PLA
EB63  C8         INY
EB64  C0 41      CPY #$41      größer $40
EB66  B0 09      BCS $EB71
EB68  CA         DEX
EB69  D0 DF      BNE $EB4A     nächste Matrix-Spalte
EB6B  38         SEC
EB6C  2E 20 91   ROL $9120
EB6F  D0 CF      BNE $EB40
EB71  6C 8F 02   JMP ($028F)   JMP $EBDC setzt Zeiger auf Kodetabelle
EB74  A4 CB      LDY $CB       Nummer der Taste
EB76  B1 F5      LDA ($F5),Y   ASCII-Wert aus Tabelle holen
EB78  AA         TAX
EB79  C4 C5      CPY $C5       mit letzter Taste vergleichen
EB7B  F0 07      BEQ $EB84
EB7D  A0 10      LDY #$10
EB7F  8C 8C 02   STY $028C     Repeat-Verzögerungszähler
EB82  D0 36      BNE $EBBA
EB84  29 7F      AND #$7F      Bit 7 löschen
EB86  2C 8A 02   BIT $028A     Repeat-Funktion für alle Tasten ?
EB89  30 16      BMI $EBA1     Bit 7 gesetzt, dann alle Tasten wiederholen
EB8B  70 49      BVS $EBD6     Bit 6 gesetzt, dann ignorieren
EB8D  C9 7F      CMP #$7F      nur folgende Tasten wiederholen
EB8F  F0 29      BEQ $EBBA
EB91  C9 14      CMP #$14      'DEL'
EB93  F0 0C      BEQ $EBA1
EB95  C9 20      CMP #$20      Blank
EB97  F0 08      BEQ $EBA1
EB99  C9 1D      CMP #$1D      Cursor RIGHT, LEFT
EB9B  F0 04      BEQ $EBA1
EB9D  C9 11      CMP #$11      Cursor DOWN, UP
EB9F  D0 35      BNE $EBD6
EBA1  AC 8C 02   LDY $028C     Repeatverzögerungszähler
EBA4  F0 05      BEQ $EBAB
EBA6  CE 8C 02   DEC $028C     runterzählen
EBA9  D0 2B      BNE $EBD6
EBAB  CE 8B 02   DEC $028B     Repeatgeschwindigkeitszähler
EBAE  D0 26      BNE $EBD6
EBB0  A0 04      LDY #$04
EBB2  8C 8B 02   STY $028B     Zähler neu setzen
EBB5  A4 C6      LDY $C6       Anzahl der Zeichen im Tastaturpuffer
EBB7  88         DEY
EBB8  10 1C      BPL $EBD6     mehr als ein Zeichen im Puffer, dann ignorieren
EBBA  A4 CB      LDY $CB
EBBC  84 C5      STY $C5
EBBE  AC 8D 02   LDY $028D
EBC1  8C 8E 02   STY $028E
EBC4  E0 FF      CPX #$FF      Tastaturkode ungültig ?
EBC6  F0 0E      BEQ $EBD6     ja, dann ignorieren
EBC8  8A         TXA
EBC9  A6 C6      LDX $C6       Anzahl der gedrückten Tasten
EBCB  EC 89 02   CPX $0289     mit Höchstzahl vergleichen
EBCE  B0 06      BCS $EBD6     größer?
EBD0  9D 77 02   STA $0277,X   Zeichen in Tastaturpuffer schreiben
EBD3  E8         INX
EBD4  86 C6      STX $C6       Anzahl erhöhen
EBD6  A9 F7      LDA #$F7      Tastatur-Matrix Abfrage auf Default
```

```
EBD8  8D 20 91     STA $9120
EBDB  60           RTS

*****************************  Prüft auf Shift, CTRL, Commodore
EBDC  AD 8D 02     LDA $028D   Flag für Shift/CTRL
EBDF  C9 03        CMP #$03
EBE1  D0 2C        BNE $EC0F   Zeiger auf Dekodiertabelle berechnen
EBE3  CD 8E 02     CMP $028E
EBE6  F0 EE        BEQ $EBD6
EBE8  AD 91 02     LDA $0291   Shift/Commodore erlaubt ?
EBEB  30 56        BMI $EC43   nein, zurück zur Dekodierung
EBED  EA ....      NOP
ECFF  .... EA      NOP
EC00  AD 05 90     LDA $9005   Shift/Commodore
EC03  49 02        EOR #$02    Umschaltung Klein/Großschreibung
EC05  8D 05 90     STA $9005
EC08  EA EA EA EA NOP's
EC0C  4C 43 EC     JMP $EC43   fertig
EC0F  0A           ASL         Flag mal 2
EC10  C9 08        CMP #$08
EC12  90 04        BCC $EC18
EC14  A9 06        LDA #$06
EC16  EA ....      NOP
EC37  .... EA      NOP
EC38  AA           TAX         Zeiger auf Tabelle
EC39  BD 46 EC     LDA $EC46,X
EC3C  85 F5        STA $F5     Adresse der Tabelle nach $F5/$F6
EC3E  BD 47 EC     LDA $EC47,X
EC41  85 F6        STA $F6
EC43  4C 74 EB     JMP $EB74   zurück zur Dekodierung

*****************************  Zeiger auf Tastatur-Dekodiertabellen
EC46  5E EC
EC48  9F EC E0 EC A3 ED 5E EC
EC50  9F EC 69 ED A3 ED 21 ED
EC58  69 ED 69 ED A3 ED

*****************************  Tastaturdekodiertabelle
EC5E  31 33
EC60  35 37 39 2B 5C 14 5F 57
EC68  52 59 49 50 2A 0D 04 41
EC70  44 47 4A 4C 3B 1D 03 01
EC78  58 56 4E 2C 2F 11 20 5A
EC80  43 42 4D 2E 01 85 02 53
EC88  46 48 4B 3A 3D 86 51 45
EC90  54 55 4F 40 5E 87 32 34
EC98  36 38 30 2D 13 88 FF

*****************************  Tastaturdekodiertabelle
EC9F  21
ECA0  23 25 27 29 DB A9 94 5F
ECA8  D7 D2 D9 C9 D0 C0 8D 04
ECB0  C1 C4 C7 CA CC 5D 9D 83
ECB8  01 D8 D6 CE 3C 3F 91 A0
ECC0  DA C3 C2 CD 3E 01 89 02
ECC8  D3 C6 C8 CB 5B 3D 8A D1
ECD0  C5 D4 D5 CF BA DE 8B 22
ECD8  24 26 28 30 DD 93 8C FF

*****************************  Tastaturdekodiertabelle
```

```
ECE0  21 23 25 27 29 A6 A8 94
ECE8  5F B3 B2 B7 A2 AF DF 8D
ECF0  04 B0 AC A5 B5 B6 5D 9D
ECF8  83 01 BD BE AA 3C 3F 91
ED00  A0 AD BC BF A7 3E 01 89
ED08  02 AE BB B4 A1 5B 3D 8A
ED10  AB B1 A3 B8 B9 A4 DE 8B
ED18  22 24 26 28 30 DC 93 8C
ED20  FF

*****************************      prüft auf Steuerzeichen
ED21  C9 0E      CMP #$0E          chr$(14)
ED23  D0 0B      BNE $ED30
ED25  A9 02      LDA #$02          Character-Generator auf Großschrift
ED27  0D 05 90   ORA $9005         umschalten
ED2A  8D 05 90   STA $9005
ED2D  4C DC E6   JMP $E6DC         fertig
ED30  C9 8E      CMP #$8E          chr$(142)
ED32  D0 0B      BNE $ED3F
ED34  A9 FD      LDA #$FD          Character-Generator auf Kleinschrift
ED36  2D 05 90   AND $9005         umschalten
ED39  8D 05 90   STA $9005
ED3C  4C DC E6   JMP $E6DC         fertig
ED3F  C9 08      CMP #$08          chr$(8)
ED41  D0 0A      BNE $ED4D
ED43  A9 80      LDA #$80
ED45  0D 91 02   ORA $0291         Shift/Commodore sperren
ED48  8D 91 02   STA $0291
ED4B  30 EF      BMI $ED3C
ED4D  C9 09      CMP #$09          chr$(9)
ED4F  D0 EB      BNE $ED3C
ED51  A9 7F      LDA #$7F
ED53  2D 91 02   AND $0291         Shift/Commodore ermöglichen
ED56  8D 91 02   STA $0291
ED59  10 E1      BPL $ED3C
ED5B  E8         INX
ED5C  B5 D9      LDA $D9,X
ED5E  09 80      ORA #$80           Bit 8 setzen
ED60  95 D9      STA $D9,X
ED62  CA         DEX
ED63  A5 D5      LDA $D5
ED65  18         CLC
ED66  4C 15 E7   JMP $E715

*****************************      Tabelle für Tastaturdekodierung
ED69  FF FF FF FF FF FF FF
ED70  FF FF FF FF FF FF FF FF
ED78  E2 9D 83 01 FF FF FF FF
ED80  FF 91 A0 FF FF FF FF EE
ED88  01 89 02 FF FF FF FF E1
ED90  FD 8A FF FF FF FF FF B0
ED98  E0 8B F2 F4 F6 FF F0 ED
EDA0  93 8C FF

*****************************      Tabelle für Tastaturdekodierung
EDA3  90 1C 9C 1F 12
EDA8  FF FF FF 06 FF 12 FF FF
EDB0  FF FF FF FF FF FF FF FF
EDB8  FF FF FF FF FF FF FF FF
EDC0  FF FF FF FF FF FF FF FF
```

```
EDC8  FF FF FF FF FF FF FF FF
EDD0  FF FF FF FF FF FF FF FF
EDD8  FF FF FF 05 9F 1E 9E 92
EDE0  FF FF FF FF

******************************   Konstanten für Videocontroller
EDE4  0C 26 16 2E                 $9000 - $900F
EDE8  00 C0 00 00 00 00 00 00
EDF0  00 00 00 1B
******************************
EDF4  4C 4F 41 44                 load
EDF8  0D 52 55 4E 0D 00           (cr) run (cr)
******************************   Tabelle der LSB der Bidschirmzeilenanfänge
EDFE  16 2C
EE00  42 58 6E 84 9A B0 C6 DC
EE08  F2 08 1E 34 4A 60 76 8C
EE10  A2 B8 CE E4

******************************   I/O-Routinen

EE14  09 40       ORA #$40        TALK senden
EE16  2C          .BYTE $2C

******************************   LISTEN senden
EE17  09 20       ORA #$20
EE19  20 60 F1    JSR $F160       IER setzen
EE1C  48          PHA
EE1D  24 94       BIT $94
EE1F  10 0A       BPL $EE2B
EE21  38          SEC
EE22  66 A3       ROR $A3
EE24  20 49 EE    JSR $EE49
EE27  46 94       LSR $94
EE29  46 A3       LSR $A3
EE2B  68          PLA
EE2C  85 95       STA $95
EE2E  20 A0 E4    JSR $E4A0       DAV = Hi ausgeben
EE31  C9 3F       CMP #$3F
EE33  D0 03       BNE $EE38
EE35  20 84 EF    JSR $EF84       NDAC Lo ausgeben
EE38  AD 1F 91    LDA $911F
EE3B  09 80       ORA #$80        ATN ausgeben
EE3D  8D 1F 91    STA $911F
EE40  20 8D EF    JSR $EF8D       PCR Bit 1 löschen
EE43  20 A0 E4    JSR $E4A0       DAV = Hi ausgeben
EE46  20 96 EF    JSR $EF96       Warten auf Interrupt von Timer 2
EE49  78          SEI             ein Byte auf IEC-Bus ausgeben (in $95)
EE4A  20 A0 E4    JSR $E4A0       DAV = Hi ausgeben
EE4D  20 B2 E4    JSR $E4B2       NRFD Hi ausgeben
EE50  4A          LSR
EE51  B0 61       BCS $EEB4       'device not present' wenn beide Hi
EE53  20 84 EF    JSR $EF84       NDAC Lo ausgeben
EE56  24 A3       BIT $A3
EE58  10 0C       BPL $EE66
EE5A  20 B2 E4    JSR $E4B2       NRFD Hi ausgeben
EE5D  4A          LSR
EE5E  90 FA       BCC $EE5A
EE60  20 B2 E4    JSR $E4B2       NRFD Hi ausgeben
EE63  4A          LSR
EE64  B0 FA       BCS $EE60
```

```
EE66  20 B2 E4    JSR $E4B2     NRFD Hi ausgeben
EE69  4A          LSR
EE6A  90 FA       BCC $EE66
EE6C  20 8D EF    JSR $EF8D     PCR Bit 1 löschen
EE6F  A9 08       LDA #$08
EE71  85 A5       STA $A5
EE73  AD 1F 91    LDA $911F
EE76  CD 1F 91    CMP $911F
EE79  D0 F8       BNE $EE73
EE7B  4A          LSR
EE7C  4A          LSR
EE7D  90 38       BCC $EEB7
EE7F  66 95       ROR $95
EE81  B0 05       BCS $EE88
EE83  20 A9 E4    JSR $E4A9
EE86  D0 03       BNE $EE8B
EE88  20 A0 E4    JSR $E4A0     DAV Hi ausgeben
EE8B  20 84 EF    JSR $EF84     NDAC Lo ausgeben
EE8E  EA          NOP
EE8F  EA          NOP
EE90  EA          NOP
EE91  EA          NOP
EE92  AD 2C 91    LDA $912C
EE95  29 DF       AND #$DF
EE97  09 02       ORA #$02
EE99  8D 2C 91    STA $912C
EE9C  C6 A5       DEC $A5
EE9E  D0 D3       BNE $EE73
EEA0  A9 04       LDA #$04
EEA2  8D 29 91    STA $9129
EEA5  AD 2D 91    LDA $912D
EEA8  29 20       AND #$20
EEAA  D0 0B       BNE $EEB7     Interrupt von Timer 2, dann time out
EEAC  20 B2 E4    JSR $E4B2     NRFD Hi ausgeben
EEAF  4A          LSR
EEB0  B0 F3       BCS $EEA5
EEB2  58          CLI
EEB3  60          RTS

EEB4  A9 80       LDA #$80      'device not present'
EEB6  2C          .BYTE #$2C
EEB7  A9 03       LDA #$03      'time out'
EEB9  20 6A FE    JSR $FE6A     STatus setzen
EEBC  58          CLI
EEBD  18          CLC
EEBE  90 49       BCC $EF09

*****************************   Sekundäradresse für LISTEN senden
EEC0  85 95       STA $95
EEC2  20 40 EE    JSR $EE40
EEC5  AD 1F 91    LDA $911F
EEC8  29 7F       AND #$7F      ATN ausgeben
EECA  8D 1F 91    STA $911F
EECD  60          RTS

*****************************   Sekundäradresse für TALK senden
EECE  85 95       STA $95
EED0  20 40 EE    JSR $EE40
EED3  78          SEI
EED4  20 A9 E4    JSR $E4A9
```

```
EED7  20 C5 EE   JSR $EEC5
EEDA  20 84 EF   JSR $EF84     NDAC Lo ausgeben
EEDD  20 B2 E4   JSR $E4B2     NRFD Hi ausgeben
EEE0  B0 FB      BCS $EEDD
EEE2  58         CLI
EEE3  60         RTS

******************************  IEC-Ausgabe
EEE4  24 94      BIT $94
EEE6  30 05      BMI $EEED
EEE8  38         SEC
EEE9  66 94      ROR $94
EEEB  D0 05      BNE $EEF2
EEED  48         PHA
EEEE  20 49 EE   JSR $EE49
EEF1  68         PLA
EEF2  85 95      STA $95
EEF4  18         CLC
EEF5  60         RTS

******************************  UNTALK senden
EEF6  20 8D EF   JSR $EF8D     PCR Bit 1 löschen
EEF9  AD 1F 91   LDA $911F
EEFC  09 80      ORA #$80      ATN ausgeben
EEFE  8D 1F 91   STA $911F
EF01  A9 5F      LDA #$5F      Kode für UNTALK
EF03  2C         .BYTE $2C

******************************  UNLISTEN senden
EF04  A9 3F      LDA #$3F      Kode für UNLISTEN
EF06  20 1C EE   JSR $EE1C
EF09  20 C5 EE   JSR $EEC5
EF0C  8A         TXA
EF0D  A2 0B      LDX #$0B
EF0F  CA         DEX
EF10  D0 FD      BNE $EF0F
EF12  AA         TAX
EF13  20 84 EF   JSR $EF84     NDAC Lo ausgeben
EF16  4C A0 E4   JMP $E4A0
******************************  IEC-Eingabe
EF19  78         SEI
EF1A  A9 00      LDA #$00
EF1C  85 A5      STA $A5
EF1E  20 84 EF   JSR $EF84     NDAC Lo ausgeben
EF21  20 B2 E4   JSR $E4B2     NRFD Hi ausgeben
EF24  90 FB      BCC $EF21
EF26  20 A0 E4   JSR $E4A0     DAV Hi ausgeben
EF29  A9 01      LDA #$01
EF2B  8D 29 91   STA $9129     Timer 2 C-H
EF2E  AD 2D 91   LDA $912D     IFR
EF31  29 20      AND #$20      Interrupt von Timer 2?
EF33  D0 07      BNE $EF3C
EF35  20 B2 E4   JSR $E4B2     NRFD Hi ausgeben
EF38  B0 F4      BCS $EF2E
EF3A  90 18      BCC $EF54
EF3C  A5 A5      LDA $A5
EF3E  F0 05      BEQ $EF45
EF40  A9 02      LDA #$02      TIMEOUT beim Lesen
EF42  4C B9 EE   JMP $EEB9     STatus setzen
```

```
EF45  20 A9 E4    JSR $E4A9
EF48  20 0C EF    JSR $EF0C
EF4B  A9 40       LDA #$40
EF4D  20 6A FE    JSR $FE6A      STatus setzen
EF50  E6 A5       INC $A5
EF52  D0 D5       BNE $EF29
EF54  A9 08       LDA #$08
EF56  85 A5       STA $A5
EF58  AD 1F 91    LDA $911F
EF5B  CD 1F 91    CMP $911F
EF5E  D0 F8       BNE $EF58
EF60  4A          LSR
EF61  90 F5       BCC $EF58
EF63  4A          LSR
EF64  66 A4       ROR $A4        ein Byte einlesen
EF66  AD 1F 91    LDA $911F
EF69  CD 1F 91    CMP $911F
EF6C  D0 F8       BNE $EF66
EF6E  4A          LSR
EF6F  B0 F5       BCS $EF66
EF71  C6 A5       DEC $A5
EF73  D0 E3       BNE $EF58
EF75  20 A9 E4    JSR $E4A9
EF78  A5 90       LDA $90
EF7A  F0 03       BEQ $EF7F
EF7C  20 0C EF    JSR $EF0C
EF7F  A5 A4       LDA $A4
EF81  58          CLI
EF82  18          CLC
EF83  60          RTS

*****************************    NDAC lo ausgeben
EF84  AD 2C 91    LDA $912C
EF87  29 FD       AND #$FD       PCR Bit 1 löschen
EF89  8D 2C 91    STA $912C
EF8C  60          RTS

*****************************    NDAC Hi ausgeben
EF8D  AD 2C 91    LDA $912C
EF90  09 02       ORA #$02       PCR Bit 1 setzen
EF92  8D 2C 91    STA $912C
EF95  60          RTS

*****************************    Warten auf Interrupt von Timer 2
EF96  A9 04       LDA #$04
EF98  8D 29 91    STA $9129
EF9B  AD 2D 91    LDA $912D
EF9E  29 20       AND #$20
EFA0  F0 F9       BEQ $EF9B
EFA2  60          RTS

*****************************    RS 232 Ausgabe
EFA3  A5 B4       LDA $B4
EFA5  F0 47       BEQ $EFEE
EFA7  30 3F       BMI $EFE8
EFA9  46 B6       LSR $B6
EFAB  A2 00       LDX #$00
EFAD  90 01       BCC $EFB0
EFAF  CA          DEX
EFB0  8A          TXA
```

```
EFB1  45 BD      EOR $BD
EFB3  85 BD      STA $BD
EFB5  C6 B4      DEC $B4
EFB7  F0 06      BEQ $EFBF
EFB9  8A         TXA
EFBA  29 20      AND #$20
EFBC  85 B5      STA $B5
EFBE  60         RTS
EFBF  A9 20      LDA #$20
EFC1  2C 94 02   BIT $0294
EFC4  F0 14      BEQ $EFDA
EFC6  30 1C      BMI $EFE4
EFC8  70 14      BVS $EFDE
EFCA  A5 BD      LDA $BD
EFCC  D0 01      BNE $EFCF
EFCE  CA         DEX
EFCF  C6 B4      DEC $B4
EFD1  AD 93 02   LDA $0293
EFD4  10 E3      BPL $EFB9
EFD6  C6 B4      DEC $B4
EFD8  D0 DF      BNE $EFB9
EFDA  E6 B4      INC $B4
EFDC  D0 F0      BNE $EFCE
EFDE  A5 BD      LDA $BD
EFE0  F0 ED      BEQ $EFCF
EFE2  D0 EA      BNE $EFCE
EFE4  70 E9      BVS $EFCF
EFE6  50 E6      BVC $EFCE
EFE8  E6 B4      INC $B4
EFEA  A2 FF      LDX #$FF
EFEC  D0 CB      BNE $EFB9
EFEE  AD 94 02   LDA $0294
EFF1  4A         LSR
EFF2  90 07      BCC $EFFB
EFF4  2C 20 91   BIT $9120
EFF7  10 1D      BPL $F016
EFF9  50 1E      BVC $F019
EFFB  A9 00      LDA #$00
EFFD  85 BD      STA $BD
EFFF  85 B5      STA $B5
F001  AE 98 02   LDX $0298
F004  86 B4      STX $B4
F006  AC 9D 02   LDY $029D
F009  CC 9E 02   CPY $029E
F00C  F0 13      BEQ $F021
F00E  B1 F9      LDA ($F9),Y
F010  85 B6      STA $B6
F012  EE 9D 02   INC $029D
F015  60         RTS

F016  A9 40      LDA #$40        DSR (Data Set Ready) fehlt
F018  2C         .BYTE $2C
F019  A9 10      LDA #$10        CTS (Clear To Send) fehlt
F01B  0D 97 02   ORA $0297
F01E  8D 97 02   STA $0297       RS 232 Status setzen
F021  A9 40      LDA #$40
F023  8D 1E 91   STA $911E
F026  60         RTS

******************************   Anzahl der RS 232 Datenbits berechnen
```

```
F027  A2 09      LDX #$09
F029  A9 20      LDA #$20
F02B  2C 93 02   BIT $0293     Kontrollwort
F02E  F0 01      BEQ $F031
F030  CA         DEX
F031  50 02      BVC $F035
F033  CA         DEX
F034  CA         DEX           X enthält Anzahl der Datenbits
F035  60         RTS

F036  A6 A9      LDX $A9
F038  D0 2E      BNE $F068
F03A  C6 A8      DEC $A8
F03C  F0 31      BEQ $F06F
F03E  30 0D      BMI $F04D
F040  A5 A7      LDA $A7
F042  45 AB      EOR $AB
F044  85 AB      STA $AB
F046  46 A7      LSR $A7
F048  66 AA      ROR $AA
F04A  60         RTS

F04B  C6 A8      DEC $A8
F04D  A5 A7      LDA $A7
F04F  F0 62      BEQ $F0B3
F051  AD 93 02   LDA $0293
F054  0A         ASL
F055  A9 01      LDA #$01
F057  65 A8      ADC $A8
F059  D0 EF      BNE $F04A
F05B  A9 90      LDA #$90
F05D  8D 1E 91   STA $911E
F060  85 A9      STA $A9
F062  A9 20      LDA #$20
F064  8D 1E 91   STA $911E
F067  60         RTS
F068  A5 A7      LDA $A7
F06A  D0 EF      BNE $F05B
F06C  85 A9      STA $A9
F06E  60         RTS

F06F  AC 9B 02   LDY $029B
F072  C8         INY
F073  CC 9C 02   CPY $029C
F076  F0 2A      BEQ $F0A2
F078  8C 9B 02   STY $029B
F07B  88         DEY
F07C  A5 AA      LDA $AA
F07E  AE 98 02   LDX $0298
F081  E0 09      CPX #$09
F083  F0 04      BEQ $F089
F085  4A         LSR
F086  E8         INX
F087  D0 F8      BNE $F081
F089  91 F7      STA ($F7),Y
F08B  A9 20      LDA #$20
F08D  2C 94 02   BIT $0294
F090  F0 B9      BEQ $F04B
F092  30 B6      BMI $F04A
F094  A5 A7      LDA $A7
```

```
F096   45 AB       EOR $AB
F098   F0 03       BEQ $F09D
F09A   70 AE       BVS $F04A
F09C   2C          .BYTE $2C
F09D   50 AB       BVC $F04A
F09F   A9 01       LDA #$01      Paraity-Fehler
F0A1   2C          .BYTE $2C
F0A2   A9 04       LDA #$04      Empfängerpuffer voll
F0A4   2C          .BYTE $2C
F0A5   A9 80       LDA #$80      Break-Befehl empfangen
F0A7   2C          .BYTE $2C
F0A8   A9 02       LDA #$02      Rahmen-Fehler
F0AA   0D 97 02    ORA $0297
F0AD   8D 97 02    STA $0297     RS 232 Status setzen
F0B0   4C 5B F0    JMP $F05B
F0B3   A5 AA       LDA $AA
F0B5   D0 F1       BNE $F0A8
F0B7   F0 EC       BEQ $F0A5
F0B9   4C 96 F7    JMP $F796     gibt 'illegal device number'

*****************************    RS 232 CKOUT
F0BC   85 9A       STA $9A       Gerätenummer
F0BE   AD 94 02    LDA $0294
F0C1   4A          LSR
F0C2   90 27       BCC $F0EB
F0C4   A9 02       LDA #$02
F0C6   2C 10 91    BIT $9110
F0C9   10 1D       BPL $F0E8
F0CB   D0 1E       BNE $F0EB
F0CD   AD 1E 91    LDA $911E
F0D0   29 30       AND #$30
F0D2   D0 F9       BNE $F0CD
F0D4   2C 10 91    BIT $9110
F0D7   70 FB       BVS $F0D4
F0D9   AD 10 91    LDA $9110
F0DC   09 02       ORA #$02
F0DE   8D 10 91    STA $9110
F0E1   2C 10 91    BIT $9110
F0E4   70 05       BVS $F0EB
F0E6   30 F9       BMI $F0E1
F0E8   20 16 F0    JSR $F016
F0EB   18          CLC
F0EC   60          RTS

*****************************    RS 232 Ausgabe
F0ED   AC 9E 02    LDY $029E
F0F0   C8          INY
F0F1   CC 9D 02    CPY $029D
F0F4   F0 F7       BEQ $F0ED
F0F6   8C 9E 02    STY $029E
F0F9   88          DEY
F0FA   91 F9       STA ($F9),Y
F0FC   2C 1E 91    BIT $911E
F0FF   50 01       BVC $F102
F101   60          RTS

F102   AD 99 02    LDA $0299
F105   8D 14 91    STA $9114
F108   AD 9A 02    LDA $029A
F10B   8D 15 91    STA $9115
```

```
F10E  A9 C0        LDA #$C0
F110  8D 1E 91     STA $911E
F113  4C EE EF     JMP $EFEE

*****************************  RS 232 CHKIN
F116  85 99        STA $99         Nummer des Eingabegeräts
F118  AD 94 02     LDA $0294
F11B  4A           LSR
F11C  90 28        BCC $F146
F11E  29 08        AND #$08
F120  F0 24        BEQ $F146
F122  A9 02        LDA #$02
F124  2C 10 91     BIT $9110
F127  10 BF        BPL $F0E8
F129  F0 19        BEQ $F144
F12B  2C 1E 91     BIT $911E
F12E  70 FB        BVS $F12B
F130  AD 10 91     LDA $9110
F133  29 FD        AND #$FD
F135  8D 10 91     STA $9110
F138  AD 10 91     LDA $9110
F13B  29 04        AND #$04
F13D  F0 F9        BEQ $F138
F13F  A9 90        LDA #$90
F141  8D 1E 91     STA $911E
F144  18           CLC
F145  60           RTS

F146  AD 1E 91     LDA $911E
F149  29 30        AND #$30
F14B  F0 F2        BEQ $F13F
F14D  18           CLC
F14E  60           RTS

*****************************  GET von RS 232
F14F  AC 9C 02     LDY $029C       Pufferzeiger
F152  CC 9B 02     CPY $029B
F155  F0 06        BEQ $F15D       Puffer leer ?
F157  B1 F7        LDA ($F7),Y     Zeichen auf Puffer holen
F159  EE 9C 02     INC $029C       Pufferzeiger incrementieren
F15C  60           RTS
F15D  A9 00        LDA #$00        Nullbyte zurückgeben
F15F  60           RTS

*****************************  Timer setzen
F160  48           PHA
F161  AD 1E 91     LDA $911E
F164  F0 0C        BEQ $F172
F166  AD 1E 91     LDA $911E
F169  29 60        AND #$60
F16B  D0 F9        BNE $F166
F16D  A9 10        LDA #$10
F16F  8D 1E 91     STA $911E
F172  68           PLA
F173  60           RTS

*****************************  I/O-Meldungen
F174  0D 49 2F 4F                  i/o
F178  20 45 52 52 4F 52 20 A3      error #
F180  0D 53 45 41 52 43 48 49      searchi
```

```
F188  4E 47 A0 46 4F 52 A0 0D  ng for
F190  50 52 45 53 53 20 50 4C  press pl
F198  41 59 20 4F 4E 20 54 41  ay on ta
F1A0  50 C5 50 52 45 53 53 20  pEpress
F1A8  52 45 43 4F 52 44 20 26  record &
F1B0  20 50 4C 41 59 20 4F 4E   play on
F1B8  20 54 41 50 C5 0D 4C 4F   tapE lo
F1C0  41 44 49 4E C7 0D 53 41  adinG sa
F1C8  56 49 4E 47 A0 0D 56 45  ving  ve
F1D0  52 49 46 59 49 4E C7 0D  rifyinG
F1D8  46 4F 55 4E 44 A0 0D 4F  found  o
F1E0  4B 8D                    k

******************************  Fehlermeldung des Betriebssytems ausgeben
F1E2  24 9D       BIT $9D       Direkt-Modus Flag
F1E4  10 0D       BPL $F1F3     Programm, dann überspringen
F1E6  B9 74 F1    LDA $F174,Y   Offset der Fehlermeldung in Y
F1E9  08          PHP
F1EA  29 7F       AND #$7F      Bit 7 löschen
F1EC  20 D2 FF    JSR $FFD2     BSOUT  ein Zeichen ausgeben
F1EF  C8          INY
F1F0  28          PLP
F1F1  10 F3       BPL $F1E6     noch weitere Buchstaben ?
F1F3  18          CLC
F1F4  60          RTS

******************************  GETIN
F1F5  A5 99       LDA $99       Eingabegerät
F1F7  D0 08       BNE $F201     nicht Tastatur ?
F1F9  A5 C6       LDA $C6       Anzahl der Zeichen im Tastaturpuffer
F1FB  F0 6D       BEQ $F26A     kein Zeichen ?
F1FD  78          SEI
F1FE  4C CF E5    JMP $E5CF     Zeichen aus Tastaturopuffer holen

F201  C9 02       CMP #$02      RS 232 ?
F203  D0 18       BNE $F21D     nein
F205  84 97       STY $97
F207  20 4F F1    JSR $F14F     Get von RS 232
F20A  A4 97       LDY $97
F20C  18          CLC
F20D  60          RTS

******************************  BASIN  Eingaberoutine
F20E  A5 99       LDA $99       Gerätenummer
F210  D0 0B       BNE $F21D     nicht Tastatur
F212  A5 D3       LDA $D3       Cursorposition
F214  85 CA       STA $CA
F216  A5 D6       LDA $D6       für Tastatureingabe setzen
F218  85 C9       STA $C9
F21A  4C 4F E6    JMP $E64F     Eingabe vom Bildschirm

F21D  C9 03       CMP #$03      Bildschirm ?
F21F  D0 09       BNE $F22A     nein

******************************  Eingabe vom Bildschirm
F221  85 D0       STA $D0
F223  A5 D5       LDA $D5
F225  85 C8       STA $C8
F227  4C 4F E6    JMP $E64F
```

```
F22A  B0 38       BCS $F264      vom IEC-Bus ?
F22C  C9 02       CMP #$02
F22E  F0 3F       BEQ $F26F      von RS 232

*****************************  Eingabe vom Band
F230  86 97       STX $97        X-Register speichern
F232  20 50 F2    JSR $F250      ein Zeichen vom Band holen
F235  B0 16       BCS $F24D      Zeichen gelesen ?
F237  48          PHA
F238  20 50 F2    JSR $F250      ein Zeichen vom Band holen
F23B  B0 0D       BCS $F24A
F23D  D0 05       BNE $F244      letztes Zeichen ?
F23F  A9 40       LDA #$40       End of File
F241  20 6A FE    JSR $FE6A      STatus setzen
F244  C6 A6       DEC $A6        Zeiger in Puffer erniedrigen
F246  A6 97       LDX $97        X-Register zurückholen
F248  68          PLA            Zeichen zurückholen
F249  60          RTS

F24A  AA          TAX
F24B  68          PLA
F24C  8A          TXA
F24D  A6 97       LDX $97
F24F  60          RTS

*****************************  ein Zeichen vom Band holen
F250  20 8A F8    JSR $F88A      Tapepufferzeiger erhöhen
F253  D0 0B       BNE $F260      noch Zeichen im Puffer ?
F255  20 C0 F8    JSR $F8C0      einen Block vom Band lesen
F258  B0 11       BCS $F26B      nein, nächsten Block vom Band lesen
F25A  A9 00       LDA #$00
F25C  85 A6       STA $A6        Pufferzeiger auf Null, Zeichen holen
F25E  F0 F0       BEQ $F250
F260  B1 B2       LDA ($B2),Y    Zeichen aus Puffer holen
F262  18          CLC
F263  60          RTS
F264  A5 90       LDA $90        Status testen
F266  F0 04       BEQ $F26C
F268  A9 0D       LDA #$0D       STatus ungleich Null, dann 'CR' geben
F26A  18          CLC
F26B  60          RTS

*****************************  IEC-Eingabe
F26C  4C 19 EF    JMP $EF19      ein Byte vom IEC-Bus holen

*****************************  RS 232 Eingabe
F26F  20 05 F2    JSR $F205      ein Byte von RS 232 holen
F272  B0 05       BCS $F279
F274  C9 00       CMP #$00
F276  F0 F7       BEQ $F26F
F278  18          CLC
F279  60          RTS

*****************************  BSOUT  Ausgabe eines Zeichens
F27A  48          PHA
F27B  A5 9A       LDA $9A        Gerätenummer für Ausgabe
F27D  C9 03       CMP #$03       Bildschirm ?
F27F  D0 04       BNE $F285      nein
F281  68          PLA
F282  4C 42 E7    JMP $E742      Ausgabe auf Bildschirm
```

```
F285  90 04       BCC $F28B
F287  68          PLA
F288  4C E4 EE    JMP $EEE4      IEC-Ausgabe

F28B  C9 02       CMP #$02       RS 232 Ausgabe
F28D  F0 2A       BEQ $F2B9

*****************************    Bandausgabe
F28F  68          PLA
F290  85 9E       STA $9E
F292  48          PHA
F293  8A          TXA
F294  48          PHA
F295  98          TYA
F296  48          PHA
F297  20 8A F8    JSR $F88A      Tapepufferzeiger erhöhen
F29A  D0 0E       BNE $F2AA      Puffer voll ?
F29C  20 E3 F8    JSR $F8E3      ja, Puffer auf Band schreiben
F29F  B0 0E       BCS $F2AF
F2A1  A9 02       LDA #$02
F2A3  A0 00       LDY #$00       Kontrollbyte fuer Datenblock in Puffer
F2A5  91 B2       STA ($B2),Y
F2A7  C8          INY
F2A8  84 A6       STY $A6        Pufferzeiger erhöhen
F2AA  A5 9E       LDA $9E        Zeichen
F2AC  91 B2       STA ($B2),Y    in Tapepuffer schreiben
F2AE  18          CLC
F2AF  68          PLA
F2B0  A8          TAY
F2B1  68          PLA
F2B2  AA          TAX
F2B3  68          PLA
F2B4  90 02       BCC $F2B8
F2B6  A9 00       LDA #$00
F2B8  60          RTS

*****************************    RS 232 Ausgabe
F2B9  68          PLA
F2BA  86 97       STX $97
F2BC  84 9E       STY $9E
F2BE  20 ED F0    JSR $F0ED      ein Zeichen auf RS 232 ausgeben
F2C1  A6 97       LDX $97
F2C3  A4 9E       LDY $9E
F2C5  18          CLC
F2C6  60          RTS

*****************************    CHKIN  Eingabegerät festlegen
F2C7  20 CF F3    JSR $F3CF      sucht logische Filenummer
F2CA  F0 03       BEQ $F2CF      gefunden ?
F2CC  4C 84 F7    JMP $F784      nein, 'file not open'
F2CF  20 DF F3    JSR $F3DF      setzt Fileparameter
F2D2  A5 BA       LDA $BA        Primäradresse
F2D4  F0 16       BEQ $F2EC      Tastatur ?
F2D6  C9 03       CMP #$03
F2D8  F0 12       BEQ $F2EC      Bildschirm ?
F2DA  B0 14       BCS $F2F0      IEC ?
F2DC  C9 02       CMP #$02
F2DE  D0 03       BNE $F2E3
F2E0  4C 16 F1    JMP $F116      RS 232
```

```
******************************   Band als Eingabegerät setzen
F2E3  A6 B9       LDX $B9        Sekundäradresse
F2E5  E0 60       CPX #$60       null
F2E7  F0 03       BEQ $F2EC      ja, lesen
F2E9  4C 8D F7    JMP $F78D      sonst  'not input file'
F2EC  85 99       STA $99        INPUT-Filenummmer
F2EE  18          CLC
F2EF  60          RTS

******************************   IEC-Bus als Eingabegerät
F2F0  AA          TAX
F2F1  20 14 EE    JSR $EE14      TALK senden
F2F4  A5 B9       LDA $B9        Sekundär-Adresse
F2F6  10 06       BPL $F2FE      positiv? =>
F2F8  20 D3 EE    JSR $EED3
F2FB  4C 01 F3    JMP $F301
F2FE  20 CE EE    JSR $EECE      Sekundäradresse für TALK senden
F301  8A          TXA
F302  24 90       BIT $90        STatus
F304  10 E6       BPL $F2EC      OK.
F306  4C 8A F7    JMP $F78A      sonst 'device not present'

******************************   CKOUT  Ausgabegerät setzen
F309  20 CF F3    JSR $F3CF      sucht log. Filenummer
F30C  F0 03       BEQ $F311      Filenummer gefunden?
F30E  4C 84 F7    JMP $F784      nein,  'file not open'
F311  20 DF F3    JSR $F3DF      Fileparameter setzen
F314  A5 BA       LDA $BA        Primär-Adresse
F316  D0 03       BNE $F31B      ungleich null ?
F318  4C 90 F7    JMP $F790      gibt 'not input file'
F31B  C9 03       CMP #$03       Bildschirm ?
F31D  F0 0F       BEQ $F32E      ja
F31F  B0 11       BCS $F332      IEC-Bus ?
F321  C9 02       CMP #$02       RS 232 ?
F323  D0 03       BNE $F328
F325  4C BC F0    JMP $F0BC      RS 232 als Ausgabegerät setzen

******************************   Band als Ausgabegerät setzen
F328  A6 B9       LDX $B9        Sekundäradresse
F32A  E0 60       CPX #$60       Null ?
F32C  F0 EA       BEQ $F318      gibt 'not input file'
F32E  85 9A       STA $9A        Nummer des Ausgabegeräts setzen
F330  18          CLC
F331  60          RTS

******************************   IEC-Bus als Ausgabegerät
F332  AA          TAX
F333  20 17 EE    JSR $EE17      LISTEN senden
F336  A5 B9       LDA $B9        Sekundär-Adresse
F338  10 05       BPL $F33F
F33A  20 C5 EE    JSR $EEC5      ATN rücksetzen
F33D  D0 03       BNE $F342
F33F  20 C0 EE    JSR $EEC0      Sekundäradresse für LISTEN senden
F342  8A          TXA
F343  24 90       BIT $90        STatus ?
F345  10 E7       BPL $F32E      OK.
F347  4C 8A F7    JMP $F78A      gibt 'device not present'

******************************   CLOSE
F34A  20 D4 F3    JSR $F3D4      sucht logische Filenummer
```

```
F34D  F0 02     BEQ $F351
F34F  18        CLC            File nicht vorhanden, fertig
F350  60        RTS
F351  20 DF F3  JSR $F3DF      Fileparameter setzen
F354  8A        TXA
F355  48        PHA
F356  A5 BA     LDA $BA        Primär-Adresse
F358  F0 57     BEQ $F3B1      Tastatur ?
F35A  C9 03     CMP #$03       Bildschirm ?
F35C  F0 53     BEQ $F3B1
F35E  B0 4E     BCS $F3AE      IEC-Bus
F360  C9 02     CMP #$02       RS 232 ?
F362  D0 29     BNE $F38D

******************************  RS 232 File schließen
F364  68        PLA
F365  20 B2 F3  JSR $F3B2      Fileeintrag in Tabelle löschen
F368  A9 7D     LDA #$7D
F36A  8D 1E 91  STA $911E
F36D  A9 06     LDA #$06
F36F  8D 10 91  STA $9110
F372  A9 EE     LDA #$EE
F374  8D 1C 91  STA $911C
F377  20 75 FE  JSR $FE75
F37A  A5 F8     LDA $F8
F37C  F0 01     BEQ $F37F
F37E  C8        INY
F37F  A5 FA     LDA $FA
F381  F0 01     BEQ $F384
F383  C8        INY
F384  A9 00     LDA #$00
F386  85 F8     STA $F8
F388  85 FA     STA $FA
F38A  4C 3C F5  JMP $F53C

******************************  Bandfile schließen
F38D  A5 B9     LDA $B9        Sekundäradresse
F38F  29 0F     AND #$0F
F391  F0 1E     BEQ $F3B1      null
F393  20 4D F8  JSR $F84D      Band-Puffer Startadresse holen
F396  A9 00     LDA #$00
F398  20 90 F2  JSR $F290      Puffer auf Band schreiben
F39B  4C CF E4  JMP $E4CF
F39E  B0 2E     BCS $F3CE
F3A0  A5 B9     LDA $B9        Sekundäradresse
F3A2  C9 62     CMP #$62
F3A4  D0 0B     BNE $F3B1
F3A6  A9 05     LDA #$05       'EOT' - Kennzeichen
F3A8  20 E7 F7  JSR $F7E7      Block auf Band schreiben
F3AB  4C B1 F3  JMP $F3B1

******************************  IEC-File schließen
F3AE  20 DA F6  JSR $F6DA      IEC-File schließen
F3B1  68        PLA
F3B2  AA        TAX
F3B3  C6 98     DEC $98        Anzahl der offenen Files erniedrigen
F3B5  E4 98     CPX $98
F3B7  F0 14     BEQ $F3CD      gleich null, dann fertig
F3B9  A4 98     LDY $98
F3BB  B9 59 02  LDA $0259,Y
```

```
F3BE  9D 59 02    STA $0259,X
F3C1  B9 63 02    LDA $0263,Y    Filedatentabelle
F3C4  9D 63 02    STA $0263,X    aktualisieren
F3C7  B9 6D 02    LDA $026D,Y
F3CA  9D 6D 02    STA $026D,X
F3CD  18          CLC
F3CE  60          RTS

*****************************   sucht logische Filenummer
F3CF  A9 00       LDA #$00
F3D1  85 90       STA $90        Status löschen
F3D3  8A          TXA            Filenummer in X
F3D4  A6 98       LDX $98        Anzahl der offenen Files
F3D6  CA          DEX
F3D7  30 15       BMI $F3EE
F3D9  DD 59 02    CMP $0259,X    sucht Eintrag in Tabelle
F3DC  D0 F8       BNE $F3D6
F3DE  60          RTS

*****************************   setzt Fileparameter
F3DF  BD 59 02    LDA $0259,X
F3E2  85 B8       STA $B8        logische Filenummer
F3E4  BD 63 02    LDA $0263,X
F3E7  85 BA       STA $BA        Gerätenummer
F3E9  BD 6D 02    LDA $026D,X
F3EC  85 B9       STA $B9        Sekundäradresse
F3EE  60          RTS

*****************************   CLALL  alle I/O-Kanäle rücksetzen
F3EF  A9 00       LDA #$00
F3F1  85 98       STA $98        Anzahl der offenen Files = Null
F3F3  A2 03       LDX #$03
F3F5  E4 9A       CPX $9A        Nummer des Ausgabegeräts
F3F7  B0 03       BCS $F3FC      kleiner als 4 ?
F3F9  20 04 EF    JSR $EF04      IEC, UNLISTEN senden
F3FC  E4 99       CPX $99        Nummer des Eingabegeräts
F3FE  B0 03       BCS $F403      kleiner als 4 ?
F400  20 F6 EE    JSR $EEF6      IEC, UNTALK senden
F403  86 9A       STX $9A
F405  A9 00       LDA #$00       Standard I/O  (Tastatur/Bildschirm)
F407  85 99       STA $99
F409  60          RTS

*****************************   OPEN
F40A  A6 B8       LDX $B8        Filenummer
F40C  D0 03       BNE $F411      ungleich null ?
F40E  4C 8D F7    JMP $F78D      gibt  'not input file' (??)
F411  20 CF F3    JSR $F3CF      sucht logische Filenummer
F414  D0 03       BNE $F419      nicht gefunden, kann neu angelegt werden
F416  4C 81 F7    JMP $F781      sonst 'file open error'
F419  A6 98       LDX $98        Anzahl der offenen Files
F41B  E0 0A       CPX #$0A       kleiner 10 ?
F41D  90 03       BCC $F422      ja
F41F  4C 7E F7    JMP $F77E      sonst 'too many files'
F422  E6 98       INC $98        Anzahl erhöhen
F424  A5 B8       LDA $B8
F426  9D 59 02    STA $0259,X
F429  A5 B9       LDA $B9        Sekundäradresse
F42B  09 60       ORA #$60
F42D  85 B9       STA $B9
```

```
F42F   9D 6D 02    STA $026D,X
F432   A5 BA       LDA $BA        Gerätenummer
F434   9D 63 02    STA $0263,X    in die entsprechenden Tabellen schreiben
F437   F0 5A       BEQ $F493      Tastatur ?
F439   C9 03       CMP #$03
F43B   F0 56       BEQ $F493      Bildschirm
F43D   90 05       BCC $F444
F43F   20 95 F4    JSR $F495      IEC-OPEN
F442   90 4F       BCC $F493
F444   C9 02       CMP #$02
F446   D0 03       BNE $F44B
F448   4C C7 F4    JMP $F4C7      RS 232 OPEN
F44B   20 4D F8    JSR $F84D      Bandpuffer Startadresse holen
F44E   B0 03       BCS $F453
F450   4C 96 F7    JMP $F796      gibt 'illegal device number'
F453   A5 B9       LDA $B9        Sekundäradresse
F455   29 0F       AND #$0F
F457   D0 1F       BNE $F478      ungleich null ?
F459   20 94 F8    JSR $F894      wartet auf PLAY-Taste
F45C   B0 36       BCS $F494
F45E   20 47 F6    JSR $F647      'searching for name' ausgeben
F461   A5 B7       LDA $B7        Länge des Filenamens
F463   F0 0A       BEQ $F46F      kein Filename, dann weiter
F465   20 67 F8    JSR $F867      gewünschten Tapeheader suchen
F468   90 18       BCC $F482
F46A   F0 28       BEQ $F494
F46C   4C 87 F7    JMP $F787      'file not found' ausgeben
F46F   20 AF F7    JSR $F7AF       nächsten Tapeheader suchen
F472   F0 20       BEQ $F494
F474   90 0C       BCC $F482
F476   B0 F4       BCS $F46C
F478   20 B7 F8    JSR $F8B7      wartet auf Record & Play Taste
F47B   B0 17       BCS $F494
F47D   A9 04       LDA #$04       Datenblock - Kennzeichen
F47F   20 E7 F7    JSR $F7E7      Block auf Band schreiben
F482   A9 BF       LDA #$BF       Zeiger auf Ende des Bandpuffers
F484   A4 B9       LDY $B9        Sekundäradresse
F486   C0 60       CPY #$60
F488   F0 07       BEQ $F491      gleich null, dann weiter
F48A   A0 00       LDY #$00
F48C   A9 02       LDA #$02       Datenheader - Kennzeichen
F48E   91 B2       STA ($B2),Y    in Puffer schreiben
F490   98          TYA
F491   85 A6       STA $A6        Zeiger in Bandpuffer
F493   18          CLC
F494   60          RTS

*******************************   File auf IEC-Bus eröffnen
F495   A5 B9       LDA $B9        Sekundäradresse
F497   30 2C       BMI $F4C5
F499   A4 B7       LDY $B7        Länge des Filenamens
F49B   F0 28       BEQ $F4C5      gleich null, dann fertig
F49D   A5 BA       LDA $BA        Geräteadresse
F49F   20 17 EE    JSR $EE17      LISTEN senden
F4A2   A5 B9       LDA $B9        Sekundäradresse
F4A4   09 F0       ORA #$F0
F4A6   20 C0 EE    JSR $EEC0      senden
F4A9   A5 90       LDA $90        Status
F4AB   10 05       BPL $F4B2      testen
F4AD   68          PLA            Rücksprungadresse löschen
```

```
F4AE   68         PLA
F4AF   4C 8A F7   JMP $F78A      gibt 'device not present'
F4B2   A5 B7      LDA $B7        Länge des Filenamens
F4B4   F0 0C      BEQ $F4C2
F4B6   A0 00      LDY #$00
F4B8   B1 BB      LDA ($BB),Y     Filenamen ausgeben
F4BA   20 E4 EE   JSR $EEE4
F4BD   C8         INY
F4BE   C4 B7      CPY $B7
F4C0   D0 F6      BNE $F4B8
F4C2   20 04 EF   JSR $EF04      UNLISTEN
F4C5   18         CLC
F4C6   60         RTS

*****************************   RS 232 OPEN
F4C7   A9 06      LDA #$06
F4C9   8D 12 91   STA $9112
F4CC   8D 10 91   STA $9110
F4CF   A9 EE      LDA #$EE
F4D1   8D 1C 91   STA $911C
F4D4   A0 00      LDY #$00
F4D6   8C 97 02   STY $0297      RS 232 Status löschen
F4D9   C4 B7      CPY $B7        Länge des "Filenamens"
F4DB   F0 0A      BEQ $F4E7
F4DD   B1 BB      LDA ($BB),Y    die ersten 4 Zeichen des Filenamens
F4DF   99 93 02   STA $0293,Y    speichern
F4E2   C8         INY
F4E3   C0 04      CPY #$04
F4E5   D0 F2      BNE $F4D9
F4E7   20 27 F0   JSR $F027      Anzahl der Datenbits berechnen
F4EA   8E 98 02   STX $0298      und speichern
F4ED   AD 93 02   LDA $0293      Kontrollregister
F4F0   29 0F      AND #$0F       Bits für Baud-Rate isolieren
F4F2   D0 00      BNE $F4F4
F4F4   0A         ASL
F4F5   AA         TAX
F4F6   BD 5A FF   LDA $FF5A,X
F4F9   0A         ASL
F4FA   A8         TAY
F4FB   BD 5B FF   LDA $FF5B,X     Zeitkonstanten für Baud-Rate
F4FE   2A         ROL
F4FF   48         PHA
F500   98         TYA
F501   69 C8      ADC #$C8
F503   8D 99 02   STA $0299
F506   68         PLA
F507   69 00      ADC #$00
F509   8D 9A 02   STA $029A
F50C   AD 94 02   LDA $0294
F50F   4A         LSR
F510   90 09      BCC $F51B
F512   AD 20 91   LDA $9120
F515   0A         ASL
F516   B0 03      BCS $F51B
F518   4C 16 F0   JMP $F016
F51B   AD 9B 02   LDA $029B
F51E   8D 9C 02   STA $029C      Pufferzeiger für RS 232 Ein/Ausgabe setzen
F521   AD 9E 02   LDA $029E
F524   8D 9D 02   STA $029D
F527   20 75 FE   JSR $FE75      Memory-Top holen
```

```
F52A  A5 F8       LDA $F8
F52C  D0 05       BNE $F533
F52E  88          DEY
F52F  84 F8       STY $F8          Zeiger für RS 232 Eingabepuffer
F531  86 F7       STX $F7
F533  A5 FA       LDA $FA
F535  D0 05       BNE $F53C
F537  88          DEY
F538  84 FA       STY $FA          Zeiger für RS 232 Ausgabepuffer
F53A  86 F9       STX $F9
F53C  38          SEC
F53D  A9 F0       LDA #$F0
F53F  4C 7B FE    JMP $FE7B        Memory-Top neu setzen

*****************************      LOAD - Routine
F542  86 C3       STX $C3          Startadresse speichern
F544  84 C4       STY $C4
F546  6C 30 03    JMP ($0330)      JMP $F549
F549  85 93       STA $93          Load/Verify-Flag
F54B  A9 00       LDA #$00
F54D  85 90       STA $90          STatus löschen
F54F  A5 BA       LDA $BA          Primäradresse
F551  D0 03       BNE $F556        ungleich null ?
F553  4C 96 F7    JMP $F796        gibt 'illegal device number'
F556  C9 03       CMP #$03         Bildschirm ?
F558  F0 F9       BEQ $F553
F55A  90 6E       BCC $F5CA        vom Band

*****************************      LOAD vom IEC-Bus
F55C  A4 B7       LDY $B7          Länge des Filnamens
F55E  D0 03       BNE $F563
F560  4C 93 F7    JMP $F793        gibt 'missing filename'
F563  20 BC E4    JSR $E4BC        'searching for name' ausgeben
F566  A9 60       LDA #$60         Sekundäradresse
F568  85 B9       STA $B9          Null für Load
F56A  20 95 F4    JSR $F495        File auf IEC-Bus eröffnen
F56D  A5 BA       LDA $BA          Gerätenummer
F56F  20 14 EE    JSR $EE14        TALK senden
F572  A5 B9       LDA $B9
F574  20 CE EE    JSR $EECE        Sekundäradresse für TALK senden
F577  20 19 EF    JSR $EF19        ein Zeichen vom IEC-Bus holen
F57A  85 AE       STA $AE          Startadresse Lo
F57C  A5 90       LDA $90          STatus
F57E  4A          LSR
F57F  4A          LSR
F580  B0 45       BCS $F5C7        Time out, dann Fehler
F582  20 19 EF    JSR $EF19        ein Zeichen vom IEC-Bus holen
F585  85 AF       STA $AF          Startadresse Hi
F587  20 C1 E4    JSR $E4C1        Startadresse für Load setzen
F58A  A9 FD       LDA #$FD
F58C  25 90       AND $90
F58E  85 90       STA $90          Status setzen
F590  20 E1 FF    JSR $FFE1        STOP-Taste gedrückt ?
F593  D0 03       BNE $F598        nein
F595  4C CB F6    JMP $F6CB        Datei auf IEC-Bus schließen
F598  20 19 EF    JSR $EF19        ein Zeichen vom IEC-Bus holen
F59B  AA          TAX
F59C  A5 90       LDA $90          Status
F59E  4A          LSR
F59F  4A          LSR
```

```
F5A0  B0 E8     BCS $F58A
F5A2  8A        TXA
F5A3  A4 93     LDY $93        Load/Verify-Flag
F5A5  F0 0C     BEQ $F5B3
F5A7  A0 00     LDY #$00
F5A9  D1 AE     CMP ($AE),Y    VERIFY
F5AB  F0 08     BEQ $F5B5
F5AD  A9 10     LDA #$10       keine übereinstimmung
F5AF  20 6A FE  JSR $FE6A      STatus setzen
F5B2  2C        .BYTE $2C
F5B3  91 AE     STA ($AE),Y    Byte abspeichern
F5B5  E6 AE     INC $AE
F5B7  D0 02     BNE $F5BB
F5B9  E6 AF     INC $AF        Zeiger erhöhen
F5BB  24 90     BIT $90
F5BD  50 CB     BVC $F58A      'EOI' ?
F5BF  20 F6 EE  JSR $EEF6      UNLISTEN senden
F5C2  20 DA F6  JSR $F6DA      CLOSE
F5C5  90 7A     BCC $F641
F5C7  4C 87 F7  JMP $F787      gibt 'file not found'

*******************************
F5CA  C9 02     CMP #$02       RS 232 ?
F5CC  D0 03     BNE $F5D1      nein
F5CE  4C B9 F0  JMP $F0B9      sonst 'illegal quantity'

*******************************  LOAD vom Band
F5D1  20 4D F8  JSR $F84D      Startadresse des Bandpuufers holen
F5D4  B0 03     BCS $F5D9
F5D6  4C 96 F7  JMP $F796      gibt 'illegal device number'
F5D9  20 94 F8  JSR $F894      wartet auf PLAY-Taste
F5DC  B0 68     BCS $F646
F5DE  20 47 F6  JSR $F647      'searching for name'
F5E1  A5 B7     LDA $B7        Länge des Filenames
F5E3  F0 09     BEQ $F5EE
F5E5  20 67 F8  JSR $F867      gewünschten Tapeheader suchen
F5E8  90 0B     BCC $F5F5
F5EA  F0 5A     BEQ $F646
F5EE  20 AF F7  JSR $F7AF      nächsten Tapeheader suchen
F5F1  F0 53     BEQ $F646
F5F3  B0 D2     BCS $F5C7
F5F5  A5 90     LDA $90
F5F7  29 10     AND #$10       STatus testen
F5F9  38        SEC
F5FA  D0 4A     BNE $F646
F5FC  E0 01     CPX #$01       Headertyp 1 = BASIC-Programm (verschieblich)
F5FE  F0 11     BEQ $F611
F600  E0 03     CPX #$03       3 = Maschinenprogramm (absolut laden)
F602  D0 DD     BNE $F5E1
F604  A0 01     LDY #$01
F606  B1 B2     LDA ($B2),Y    Startadresse low
F608  85 C3     STA $C3
F60A  C8        INY
F60B  B1 B2     LDA ($B2),Y    Startadresse high
F60D  85 C4     STA $C4
F60F  B0 04     BCS $F615
F611  A5 B9     LDA $B9        Sekundäradresse
F613  D0 EF     BNE $F604      ungleich null, dann absolut laden
F615  A0 03     LDY #$03
F617  B1 B2     LDA ($B2),Y
```

```
F619  A0 01     LDY #$01      Endadresse
F61B  F1 B2     SBC ($B2),Y
F61D  AA        TAX
F61E  A0 04     LDY #$04
F620  B1 B2     LDA ($B2),Y
F622  A0 02     LDY #$02      minus Startadresse
F624  F1 B2     SBC ($B2),Y
F626  A8        TAY
F627  18        CLC           gleich Startadresse
F628  8A        TXA
F629  65 C3     ADC $C3
F62B  85 AE     STA $AE       Programmlänge + Startadresse
F62D  98        TYA
F62E  65 C4     ADC $C4       gleich Endadresse
F630  85 AF     STA $AF
F632  A5 C3     LDA $C3
F634  85 C1     STA $C1       Startadresse nach $C1/$C2
F636  A5 C4     LDA $C4
F638  85 C2     STA $C2
F63A  20 6A F6  JSR $F66A     'loading/verifying' ausgeben
F63D  20 C9 F8  JSR $F8C9     Programm vom Band laden
F640  24        .BYTE $24
F641  18        CLC
F642  A6 AE     LDX $AE       Endadresse nach X/Y
F644  A4 AF     LDY $AF
F646  60        RTS

*****************************  'searching for filename' ausgeben
F647  A5 9D     LDA $9D       Direkt-Modus testen
F649  10 1E     BPL $F669     nein, dann übergehen
F64B  A0 0C     LDY #$0C      Offset für 'searching'
F64D  20 E6 F1  JSR $F1E6     Meldung ausgeben
F650  A5 B7     LDA $B7       Länge des Filenamens
F652  F0 15     BEQ $F669     gleich null, dann fertig
F654  A0 17     LDY #$17      Offset für 'for'
F656  20 E6 F1  JSR $F1E6     Meldung ausgeben
F659  A4 B7     LDY $B7       Länge des Filenamens
F65B  F0 0C     BEQ $F669     gleich null, dann fertig
F65F  B1 BB     LDA ($BB),Y   Filenamen
F561  20 D2 FF  JSR $FFD2     ausgeben
F664  C8        INY
F665  C4 B7     CPY $B7
F667  D0 F6     BNE $F65F
F669  60        RTS

*******************************  'loading/verifying' ausgeben
F66A  A0 49     LDY #$49      Offset für 'loading'
F66C  A5 93     LDA $93       Load/Verify-Flag
F66E  F0 02     BEQ $F672     Load, dann ausgeben
F670  A0 59     LDY #$59      Offset für 'verifying'
F672  4C E2 F1  JMP $F1E2     Meldung ausgeben

*******************************  SAVE-Routine
F675  86 AE     STX $AE       Endadresse
F677  84 AF     STY $AF
F679  AA        TAX
F67A  B5 00     LDA $00,X
F67C  85 C1     STA $C1       Programmstartadresse
F67E  B5 01     LDA $01,X
F680  85 C2     STA $C2
```

```
F682  6C 32 03    JMP ($0332)    JMP $F685
F685  A5 BA       LDA $BA        Primäradresse
F687  D0 03       BNE $F68C      nicht null ?
F689  4C 96 F7    JMP $F796      gibt 'illegal device number'
F68C  C9 03       CMP #$03
F68E  F0 F9       BEQ $F689      Bildschirm, dann Fehler
F690  90 5F       BCC $F6F1      Band

******************************           Speichern auf IEC-Bus
F692  A9 61       LDA #$61       Sekundäradresse für SAVE
F694  85 B9       STA $B9
F696  A4 B7       LDY $B7        Länge des Filenamens
F698  D0 03       BNE $F69D      nicht null ?
F69A  4C 93 F7    JMP $F793      sonst 'missing filename'
F69D  20 95 F4    JSR $F495      Filenamen auf IEC-Bus ausgeben
F6A0  20 28 F7    JSR $F728      'SAVING Name' ausgeben
F6A3  A5 BA       LDA $BA        Primär-Adresse
F6A5  20 17 EE    JSR $EE17      LISTEN senden
F6A8  A5 B9       LDA $B9        Sekundäradresse
F6AA  20 C0 EE    JSR $EEC0      für LISTEN senden
F6AD  A0 00       LDY #$00
F6AF  20 D2 FB    JSR $FBD2      Startadresse nach $AC/$AD
F6B2  A5 AC       LDA $AC        Startadresse low
F6B4  20 E4 EE    JSR $EEE4      senden
F6B7  A5 AD       LDA $AD        high
F6B9  20 E4 EE    JSR $EEE4      senden
F6BC  20 11 FD    JSR $FD11      Endadresse schon erreicht ?
F6BF  B0 16       BCS $F6D7      ja
F6C1  B1 AC       LDA ($AC),Y    Programm-Bytes
F6C3  20 E4 EE    JSR $EEE4      senden
F6C6  20 E1 FF    JSR $FFE1      STOP-Taste gedrückt ?
F6C9  D0 07       BNE $F6D2      nein
F6CB  20 DA F6    JSR $F6DA      IEC-Bus Kanal schließen
F6CE  A9 00       LDA #$00
F6D0  38          SEC            Flag für 'break' setzen
F6D1  60          RTS
F6D2  20 1B FD    JSR $FD1B      laufende Adresse erhöhen
F6D5  D0 E5       BNE $F6BC
F6D7  20 04 EF    JSR $EF04      UNLISTEN senden
F6DA  24 B9       BIT $B9        Sekundäradresse
F6DC  30 11       BMI $F6EF
F6DE  A5 BA       LDA $BA        Primäradresse
F6E0  20 17 EE    JSR $EE17      LISTEN senden
F6E3  A5 B9       LDA $B9        Sekundäradresse
F6E5  29 EF       AND #$EF
F6E7  09 E0       ORA #$E0
F6E9  20 C0 EE    JSR $EEC0      ausgeben
F6EC  20 04 EF    JSR $EF04      UNLISTEN senden
F6EF  18          CLC
F6F0  60          RTS

F6F1  C9 02       CMP #$02       Gerätenummer durch 2
F6F3  D0 03       BNE $F6F8
F6F5  4C B9 F0    JMP $F0B9      RS 232, dann 'illegal device number'

******************************           SAVE auf Band
F6F8  20 4D F8    JSR $F84D      Startadresse des Bandpuffers holen
F6FB  90 8C       BCC $F689
F6FD  20 B7 F8    JSR $F8B7      wartet auf PLAY & RECORD-Taste
F700  B0 25       BCS $F727
```

```
F702  20 28 F7    JSR $F728      'saving' ausgeben
F705  A2 03       LDX #$03       Header-Typ 3 = Maschinenprogramm (absolut)
F707  A5 B9       LDA $B9        Sekundäradresse
F709  29 01       AND #$01       Bit 0 gesetzt (1 oder 3)
F70B  D0 02       BNE $F70F      ja, dann Maschinenprogramm
F70D  A2 01       LDX #$01       Header-Typ 1 = BASIC-Programm (verschieblich
F70F  8A          TXA
F710  20 E7 F7    JSR $F7E7      Header auf Band schreiben
F713  B0 12       BCS $F727      Aussprung bei STOP-Taste
F715  20 E6 F8    JSR $F8E6      Programm auf Band schreiben
F718  B0 0D       BCS $F727      Aussprung bei STOP-Taste
F71A  A5 B9       LDA $B9        Sekundäradresse
F71C  29 02       AND #$02       Bit 1 gesetzt (2 oder 3)
F71E  F0 06       BEQ $F726      nein, dann fertig
F720  A9 05       LDA #$05       'EOT' - Kennzeichen
F722  20 E7 F7    JSR $F7E7      Block auf Band schreiben
F725  24          .BYTE $2C
F726  18          CLC
F727  60          RTS

******************************   'saving' ausgeben
F728  A5 9D       LDA $9D        Direkt-Modus testen
F72A  10 FB       BPL $F727      Programm, dann fertig
F72C  A0 51       LDY #$51       Offset für 'saving'
F72E  20 E6 F1    JSR $F1E6      Meldung ausgeben
F731  4C 59 F6    JMP $F659      Filenamen ausgeben

******************************   Uhrzeit erhöhen und STOP-Taste abfragen
F734  A2 00       LDX #$00
F736  E6 A2       INC $A2
F738  D0 06       BNE $F740
F73A  E6 A1       INC $A1        Zeit erhöhen
F73C  D0 02       BNE $F740
F73E  E6 A0       INC $A0
F740  38          SEC
F741  A5 A2       LDA $A2
F743  E9 01       SBC #$01
F745  A5 A1       LDA $A1
F747  E9 1A       SBC #$1A
F749  A5 A0       LDA $A0
F74B  E9 4F       SBC #$4F
F74D  90 06       BCC $F755      noch nicht 24 h ?
F74F  86 A0       STX $A0
F751  86 A1       STX $A1        Uhr auf Null stellen
F753  86 A2       STX $A2
F755  AD 2F 91    LDA $912F
F758  CD 2F 91    CMP $912F
F75B  D0 F8       BNE $F755
F75D  85 91       STA $91        Flag für RUN/STOP-Taste setzen
F75F  60          RTS

******************************   TIME holen
F760  78          SEI
F761  A5 A2       LDA $A2
F763  A6 A1       LDX $A1        TIME nach A/X/Y
F765  A4 A0       LDY $A0

******************************   TIME setzen
F767  78          SEI
F768  85 A2       STA $A2
```

```
F76A  86 A1       STX $A1        A/X/Y nach TIME speichern
F76C  84 A0       STY $A0
F76E  58          CLI
F76F  60          RTS

****************************     STOP-Taste abfragen
F770  A5 91       LDA $91        Flag für STOP-Taste
F772  C9 FE       CMP #$FE       testen
F774  D0 07       BNE $F77D
F776  08          PHP
F777  20 CC FF    JSR $FFCC      CLRCH  I/O-Kanäle rücksetzen
F77A  85 C6       STA $C6
F77C  28          PLP
F77D  60          RTS

****************************     Meldungen des Betriebssystems ausgeben
F77E  A9 01       LDA #$01
F780  2C          .BYTE $2C
F781  A9 02       LDA #$02
F783  2C          .BYTE $2C
F784  A9 03       LDA #$03
F786  2C          .BYTE $2C
F787  A9 04       LDA #$04
F789  2C          .BYTE $2C
F78A  A9 05       LDA #$05
F78C  2C          .BYTE $2C
F784  A9 06       LDA #$06
F78F  2C          .BYTE $2C
F784  A9 07       LDA #$07
F792  2C          .BYTE $2C
F784  A9 08       LDA #$08
F795  2C          .BYTE $2C
F784  A9 09       LDA #$09
F798  48          PHA
F799  20 CC FF    JSR $FFCC      CLRCH  I/O-Kanäle rücksetzen
F79C  A0 00       LDY #$00
F79E  24 9D       BIT $9D        prüft auf Bit 6 im Pgr/Direkt-Flag
F7A0  50 0A       BVC $F7AC
F7A2  20 E6 F1    JSR $F1E6      'I/O-ERROR# X'  X=0..9
F7A5  68          PLA
F7A6  48          PHA
F7A7  09 30       ORA #$30       Zahl nach ASCII wandlen
F7A9  20 D2 FF    JSR $FFD2      und ausgeben
F7AC  68          PLA
F7AD  38          SEC
F7AE  60          RTS

****************************     Programm Header vom Band lesen
F7AF  A5 93       LDA $93        Load/Verify-Flag retten
F7B1  48          PHA
F7B2  20 C0 F8    JSR $F8C0      einen Block vom Band lesen
F7B5  68          PLA
F7B6  85 93       STA $93
F7B8  B0 2C       BCS $F7E6
F7BA  A0 00       LDY #$00
F7BC  B1 B2       LDA ($B2),Y    Kontrollbyte
F7BE  C9 05       CMP #$05       'EOT'-Block ?
F7C0  F0 24       BEQ $F7E6
F7C2  C9 01       CMP #$01       BASIC-Programm ?
F7C4  F0 08       BEQ $F7CE
```

```
F7C6  C9 03       CMP #$03      Maschinenprogramm ?
F7C8  F0 04       BEQ $F7CE
F7CA  C9 04       CMP #$04      Datenheader ?
F7CC  D0 E1       BNE $F7AF     von vorne
F7CE  AA          TAX
F7CF  24 9D       BIT $9D       prüft auf Bit 6 im Pgr/Direkt-Flag
F7D1  10 11       BPL $F7E4     Direkt-Modus ?
F7D3  A0 63       LDY #$63      Offset für 'found'
F7D5  20 E6 F1    JSR $F1E6     Meldung ausgeben
F7D8  A0 05       LDY #$05
F7DA  B1 B2       LDA ($B2),Y   Filenamen
F7DC  20 D2 FF    JSR $FFD2     ausgeben
F7DF  C8          INY
F7E0  C0 15       CPY #$15
F7E2  D0 F6       BNE $F7DA
F7E4  18          CLC
F7E5  88          DEY
F7E6  60          RTS

******************************  Header generieren und auf Band schreiben
F7E7  85 9E       STA $9E       Header-Typ
F7E9  20 4D F8    JSR $F84D     Bandpufferadresse holen
F7EC  90 5E       BCC $F84C
F7EE  A5 C2       LDA $C2
F7F0  48          PHA           Startadresse
F7F1  A5 C1       LDA $C1
F7F3  48          PHA
F7F4  A5 AF       LDA $AF
F7F6  48          PHA           Endadresse
F7F7  A5 AE       LDA $AE
F7F9  48          PHA
F7FA  A0 BF       LDY #$BF      Pufferlänge minus 1
F7FC  A9 20       LDA #$20
F7FE  91 B2       STA ($B2),Y   Bandpuffer löschen
F800  88          DEY
F801  D0 FB       BNE $F7FE
F803  A5 9E       LDA $9E       Haedertyp
F805  91 B2       STA ($B2),Y
F807  C8          INY
F808  A5 C1       LDA $C1
F80A  91 B2       STA ($B2),Y
F80C  C8          INY
F80D  A5 C2       LDA $C2       Startadresse
F80F  91 B2       STA ($B2),Y
F811  C8          INY
F812  A5 AE       LDA $AE
F814  91 B2       STA ($B2),Y
F816  C8          INY
F817  A5 AF       LDA $AF       Endadresse
F819  91 B2       STA ($B2),Y
F81B  C8          INY
F81C  84 9F       STY $9F
F81E  A0 00       LDY #$00
F820  84 9E       STY $9E
F822  A4 9E       LDY $9E
F824  C4 B7       CPY $B7
F826  F0 0C       BEQ $F834
F828  B1 BB       LDA ($BB),Y   Filenamen
F82A  A4 9F       LDY $9F
F82C  91 B2       STA ($B2),Y
```

```
F82E  E6 9E       INC $9E
F830  E6 9F       INC $9F
F832  D0 EE       BNE $F822
F834  20 54 F8    JSR $F854     Start und Endadresse auf Bandpuffer
F837  A9 69       LDA #$69
F839  85 AB       STA $AB
F83B  20 EA F8    JSR $F8EA     Puffer auf Band schreiben
F83E  A8          TAY
F83F  68          PLA
F840  85 AE       STA $AE
F842  68          PLA
F843  85 AF       STA $AF
F845  68          PLA            Adressen zurückholen
F846  85 C1       STA $C1
F848  68          PLA
F849  85 C2       STA $C2
F84B  98          TYA
F84C  60          RTS

*****************************   Startadresse des Bandpuffers holen
F84D  A6 B2       LDX $B2
F84F  A4 B3       LDY $B3       X/X zeigt auf Bandpuffer
F851  C0 02       CPY #$02
F853  60          RTS

*****************************
F854  20 4D F8    JSR $F84D     Startadresse des Bandpuffers holen
F857  8A          TXA            *Pufferende nach $AE/$AF
F858  85 C1       STA $C1
F85A  18          CLC           Startadresse nach $C1/$C2
F85B  69 C0       ADC #$C0
F85D  85 AE       STA $AE
F85F  98          TYA           Endadresse nach $AE/$AF
F860  85 C2       STA $C2
F862  69 00       ADC #$00
F864  85 AF       STA $AF
F866  60          RTS

*****************************   Bandheader nach Namen suchen
F867  20 AF F7    JSR $F7AF     nächsten Bandheader suchen
F86A  B0 1D       BCS $F889     'EOT' ?
F86C  A0 05       LDY #$05      Offset für Filenamen im Header
F86E  84 9F       STY $9F
F870  A0 00       LDY #$00
F872  84 9E       STY $9E
F874  C4 B7       CPY $B7       Länge des gesuchten Filenamens
F876  F0 10       BEQ $F888
F878  B1 BB       LDA ($BB),Y   gesuchten Filenamen
F87A  A4 9F       LDY $9F
F87C  D1 B2       CMP ($B2),Y   mit Filenamen auf Band
F87E  D0 E7       BNE $F867     vergleichen
F880  E6 9E       INC $9E
F882  E6 9F       INC $9F       Zähler erhöhen
F884  A4 9E       LDY $9E
F886  D0 EC       BNE $F874
F888  18          CLC
F889  60          RTS

*****************************   Bandpufferzeiger erhöhen
F88A  20 4D F8    JSR $F84D     Bandpufferadresse holen
```

```
F88D  E6 A6       INC $A6       Zeiger erhöhen
F88F  A4 A6       LDY $A6
F891  C0 C0       CPY #$C0      mit Maximalwert vergleichen
F893  60          RTS

*****************************   Wartet auf Play-Taste
F894  20 AB F8    JSR $F8AB     fragt Bandtaste ab
F897  F0 1C       BEQ $F8B5     gedrückt, dann fertig
F899  A0 1B       LDY #$1B      Offset für 'press play on tape'
F89B  20 E6 F1    JSR $F1E6     Meldung ausgeben
F89E  20 4B F9    JSR $F94B     testet auf STOP-Taste
F8A1  20 AB F8    JSR $F8AB     fragt Bandtaste ab
F8A4  D0 F8       BNE $F89E
F8A6  A0 6A       LDY #$6A      Offset für 'ok'
F8A8  4C E6 F1    JMP $F1E6     Meldung ausgeben

*****************************   Abfrage auf Bandtaste
F8AB  A9 40       LDA #$40
F8AD  2C 1F 91    BIT $911F
F8B0  D0 03       BNE $F8B5     testet Bandtaste
F8B2  2C 1F 91    BIT $911F
F8B5  18          CLC
F8B6  60          RTS

*****************************   Wartet auf Record & Play-Taste
F8B7  20 AB F8    JSR $F8AB     fragt Bandtaste ab
F8BA  F0 F9       BEQ $F8B5     gedrückt, dann fertig
F8BC  A0 2E       LDY #$2E      Nummer für 'press record & play on tape'
F8BE  D0 DB       BNE $F89B     weiter wie oben

*****************************   Block vom Band lesen
F8C0  A9 00       LDA #$00
F8C2  85 90       STA $90       Status
F8C4  85 93       STA $93       und Load/Verify-Flag löschen
F8C6  20 54 F8    JSR $F854     Bandpufferadresse holen

*****************************   Programm vom Band laden
F8C9  20 94 F8    JSR $F894     wartet auf PLAY-Taste
F8CC  B0 1F       BCS $F8ED
F8CE  78          SEI
F8CF  A9 00       LDA #$00
F8D1  85 AA       STA $AA
F8D3  85 B4       STA $B4
F8D5  85 B0       STA $B0       Arbeitsspeicher für IRQ-Routine löschen
F8D7  85 9E       STA $9E
F8D9  85 9F       STA $9F
F8DB  85 9C       STA $9C
F8DD  A9 82       LDA #$82      Konstante für Timing
F8DF  A2 0E       LDX #$0E      Nummer des IRQ-Vektors
F8E1  D0 11       BNE $F8F4

*****************************   Bandpuffer auf Band schreiben
F8E3  20 54 F8    JSR $F854     Bandpufferadresse holen
F8E6  A9 14       LDA #$14      Checksumme
F8E8  85 AB       STA $AB

*****************************   Block bzw. Programm auf Band schreiben
F8EA  20 B7 F8    JSR $F8B7     wartet auf Record & Play -Taste
F8ED  B0 68       BCS $F957
F8EF  78          SEI
```

```
F8F0  A9 A0      LDA #$A0     Konstante für Timing
F8F2  A2 08      LDX #$08     Nummer des IRQ-Vektors
F8F4  A0 7F      LDY #$7F
F8F6  8C 2E 91   STY $912E
F8F9  8D 2E 91   STA $912E
F8FC  20 60 F1   JSR $F160    IER setzen
F8FF  AD 14 03   LDA $0314
F902  8D 9F 02   STA $029F    IRQ-Vektor
F905  AD 15 03   LDA $0315
F908  8D A0 02   STA $02A0    nach $029F/$02A0
F90B  20 FB FC   JSR $FCFB    IRQ-Vektor für Band I/O setzen
F90E  A9 02      LDA #$02
F910  85 BE      STA $BE      Anzahl der Blocks
F912  20 DB FB   JSR $FBDB    serielle Ausgabe vorbereiten, Bitzähler setzen
F915  AD 1C 91   LDA $911C
F918  29 FD      AND #$FD     Bandmotor einschalten
F91A  09 0C      ORA #$0C
F91C  8D 1C 91   STA $911C
F91F  85 C0      STA $C0      Flag für Bandmotor
F921  A2 FF      LDX #$FF
F923  A0 FF      LDY #$FF
F925  88         DEY
F926  D0 FD      BNE $F925    Verzögerung für Bandhochlaufzeit
F928  CA         DEX
F929  D0 F8      BNE $F923
F92B  8D 29 91   STA $9129
F92E  58         CLI          Interrupt für Band I/O freigeben
*****************************  wartet auf I/O-Abschluß
F92F  AD A0 02   LDA $02A0    IRQ-Vektor wieder auf Standard-Wert ?
F932  CD 15 03   CMP $0315
F935  18         CLC
F936  F0 1F      BEQ $F957    ja, dann fertig
F938  20 4B F9   JSR $F94B    Test auf STOP-Taste
F93B  AD 2D 91   LDA $912D
F93E  29 40      AND #$40
F940  F0 ED      BEQ $F92F
F942  AD 14 91   LDA $9114
F945  20 34 F7   JSR $F734    Uhr weiter stellen
F948  4C 2F F9   JMP $F92F    weiter warten

*****************************  Testet auf STOP-Taste
F94B  20 E1 FF   JSR $FFE1    STOP-Taste gedrückt ?
F94E  18         CLC
F94F  D0 0B      BNE $F95C    nein, dann Rückkehr
F951  20 CF FC   JSR $FCCF
F954  38         SEC
F955  68         PLA
F956  68         PLA
F957  A9 00      LDA #$00
F959  8D A0 02   STA $02A0    Flag für Standard IRQ
F95C  60         RTS

*****************************  Band für Lesen vorbereiten
F95D  86 B1      STX $B1
F95F  A5 B0      LDA $B0
F961  0A         ASL
F962  0A         ASL
F963  18         CLC
F964  65 B0      ADC $B0
F966  18         CLC
```

```
F967  65 B1       ADC $B1
F969  85 B1       STA $B1
F96B  A9 00       LDA #$00
F96D  24 B0       BIT $B0
F96F  30 01       BMI $F972
F971  2A          ROL
F972  06 B1       ASL $B1
F974  2A          ROL
F975  06 B1       ASL $B1
F977  2A          ROL
F978  AA          TAX
F979  AD 28 91    LDA $9128
F97C  C9 15       CMP #$15
F97E  90 F9       BCC $F979
F980  65 B1       ADC $B1
F982  8D 24 91    STA $9124
F985  8A          TXA
F986  6D 29 91    ADC $9129
F989  8D 25 91    STA $9125
F98C  58          CLI
F98D  60          RTS

****************************  Interrupt-Routine für Band lesen
F98E  AE 29 91    LDX $9129     Timer high
F991  A0 FF       LDY #$FF
F993  98          TYA
F994  ED 28 91    SBC $9128
F997  EC 29 91    CPX $9129
F99A  D0 F2       BNE $F98E
F99C  86 B1       STX $B1
F99E  AA          TAX
F99F  8C 28 91    STY $9128
F9A2  8C 29 91    STY $9129
F9A5  98          TYA
F9A6  E5 B1       SBC $B1
F9A8  86 B1       STX $B1
F9AA  4A          LSR
F9AB  66 B1       ROR $B1
F9AD  4A          LSR
F9AE  66 B1       ROR $B1
F9B0  A5 B0       LDA $B0
F9B2  18          CLC
F9B3  69 3C       ADC #$3C
F9B5  2C 21 91    BIT $9121
F9B8  C5 B1       CMP $B1
F9BA  B0 4A       BCS $FA06
F9BC  A6 9C       LDX $9C
F9BE  F0 03       BEQ $F9C3
F9C0  4C AD FA    JMP $FAAD

F9C3  A6 A3       LDX $A3
F9C5  30 1B       BMI $F9E2
F9C7  A2 00       LDX #$00
F9C9  69 30       ADC #$30
F9CB  65 B0       ADC $B0
F9CD  C5 B1       CMP $B1
F9CF  B0 1C       BCS $F9ED
F9D1  E8          INX
F9D2  69 26       ADC #$26
F9D4  65 B0       ADC $B0
```

```
F9D6  C5 B1       CMP $B1
F9D8  B0 17       BCS $F9F1
F9DA  69 2C       ADC #$2C
F9DC  65 B0       ADC $B0
F9DE  C5 B1       CMP $B1
F9E0  90 03       BCC $F9E5
F9E2  4C 60 FA    JMP $FA60

F9E5  A5 B4       LDA $B4
F9E7  F0 1D       BEQ $FA06
F9E9  85 A8       STA $A8
F9EB  D0 19       BNE $FA06
F9ED  E6 A9       INC $A9
F9EF  B0 02       BCS $F9F3
F9F1  C6 A9       DEC $A9
F9F3  38          SEC
F9F4  E9 13       SBC #$13
F9F6  E5 B1       SBC $B1
F9F8  65 92       ADC $92
F9FA  85 92       STA $92
F9FC  A5 A4       LDA $A4
F9FE  49 01       EOR #$01
FA00  85 A4       STA $A4
FA02  F0 21       BEQ $FA25
FA04  86 D7       STX $D7
FA06  A5 B4       LDA $B4
FA08  F0 18       BEQ $FA22
FA0A  2C 2D 91    BIT $912D
FA0D  50 13       BVC $FA22
FA0F  A9 00       LDA #$00
FA11  85 A4       STA $A4
FA13  A5 A3       LDA $A3
FA15  10 30       BPL $FA47
FA17  30 C9       BMI $F9E2
FA19  A2 A6       LDX #$A6
FA1B  20 5D F9    JSR $F95D
FA1E  A5 9B       LDA $9B
FA20  D0 C3       BNE $F9E5
FA22  4C 56 FF    JMP $FF56       Rückkehr vom Interrupt

FA25  A5 92       LDA $92
FA27  F0 07       BEQ $FA30
FA29  30 03       BMI $FA2E
FA2B  C6 B0       DEC $B0
FA2D  2C          .BYTE $2C
FA2E  E6 B0       INC $B0
FA30  A9 00       LDA #$00
FA32  85 92       STA $92
FA34  E4 D7       CPX $D7
FA36  D0 0F       BNE $FA47
FA38  8A          TXA
FA39  D0 AA       BNE $F9E5
FA3B  A5 A9       LDA $A9
FA3D  30 C7       BMI $FA06
FA3F  C9 10       CMP #$10
FA41  90 C3       BCC $FA06
FA43  85 96       STA $96
FA45  B0 BF       BCS $FA06
FA47  8A          TXA
FA48  45 9B       EOR $9B
```

```
FA4A   85 9B       STA $9B
FA4C   A5 B4       LDA $B4
FA4E   F0 D2       BEQ $FA22
FA50   C6 A3       DEC $A3
FA52   30 C5       BMI $FA19
FA54   46 D7       LSR $D7
FA56   66 BF       ROR $BF
FA58   A2 DA       LDX #$DA
FA5A   20 5D F9    JSR $F95D
FA5D   4C 56 FF    JMP $FF56      Rückkehr vom Interrupt

FA60   A5 96       LDA $96
FA62   F0 04       BEQ $FA68
FA64   A5 B4       LDA $B4
FA66   F0 04       BEQ $FA6C
FA68   A5 A3       LDA $A3
FA6A   10 85       BPL $F9F1
FA6C   46 B1       LSR $B1
FA6E   A9 93       LDA #$93
FA70   38          SEC
FA71   E5 B1       SBC $B1
FA73   65 B0       ADC $B0
FA75   0A          ASL
FA76   AA          TAX
FA77   20 5D F9    JSR $F95D
FA7A   E6 9C       INC $9C
FA7C   A5 B4       LDA $B4
FA7E   D0 11       BNE $FA91
FA80   A5 96       LDA $96
FA82   F0 26       BEQ $FAAA
FA84   85 A8       STA $A8
FA86   A9 00       LDA #$00
FA88   85 96       STA $96
FA8A   A9 C0       LDA #$C0
FA8C   8D 2E 91    STA $912E
FA8F   85 B4       STA $B4
FA91   A5 96       LDA $96
FA93   85 B5       STA $B5
FA95   F0 09       BEQ $FAA0
FA97   A9 00       LDA #$00
FA99   85 B4       STA $B4
FA9B   A9 40       LDA #$40
FA9D   8D 2E 91    STA $912E
FAA0   A5 BF       LDA $BF
FAA2   85 BD       STA $BD
FAA4   A5 A8       LDA $A8
FAA6   05 A9       ORA $A9
FAA8   85 B6       STA $B6
FAAA   4C 56 FF    JMP $FF56      Rückkehr vom Interrupt

FAAD   20 DB FB    JSR $FBDB
FAB0   85 9C       STA $9C
FAB2   A2 DA       LDX #$DA
FAB4   20 5D F9    JSR $F95D
FAB7   A5 BE       LDA $BE
FAB9   F0 02       BEQ $FABD
FABB   85 A7       STA $A7
FABD   A9 0F       LDA #$0F
FABF   24 AA       BIT $AA
FAC1   10 17       BPL $FADA
```

```
FAC3  A5 B5      LDA $B5
FAC5  D0 0C      BNE $FAD3
FAC7  A6 BE      LDX $BE
FAC9  CA         DEX
FACA  D0 0B      BNE $FAD7
FACC  A9 08      LDA #$08      'long block error'
FACE  20 6A FE   JSR $FE6A     STatus setzen
FAD1  D0 04      BNE $FAD7
FAD3  A9 00      LDA #$00
FAD5  85 AA      STA $AA
FAD7  4C 56 FF   JMP $FF56     Rückkehr vom Interrupt

FADA  70 31      BVS $FB0D
FADC  D0 18      BNE $FAF6
FADE  A5 B5      LDA $B5
FAE0  D0 F5      BNE $FAD7
FAE2  A5 B6      LDA $B6
FAE4  D0 F1      BNE $FAD7
FAE6  A5 A7      LDA $A7
FAE8  4A         LSR
FAE9  A5 BD      LDA $BD
FAEB  30 03      BMI $FAF0
FAED  90 18      BCC $FB07
FAEF  18         CLC
FAF0  B0 15      BCS $FB07
FAF2  29 0F      AND #$0F
FAF4  85 AA      STA $AA
FAF6  C6 AA      DEC $AA
FAF8  D0 DD      BNE $FAD7
FAFA  A9 40      LDA #$40
FAFC  85 AA      STA $AA
FAFE  20 D2 FB   JSR $FBD2
FB01  A9 00      LDA #$00
FB03  85 AB      STA $AB
FB05  F0 D0      BEQ $FAD7
FB07  A9 80      LDA #$80
FB09  85 AA      STA $AA
FB0B  D0 CA      BNE $FAD7
FB0D  A5 B5      LDA $B5
FB0F  F0 0A      BEQ $FB1B
FB11  A9 04      LDA #$04      'short block error'
FB13  20 6A FE   JSR $FE6A     STatus setzen
FB16  A9 00      LDA #$00
FB18  4C 97 FB   JMP $FB97
FB1B  20 11 FD   JSR $FD11
FB1E  90 03      BCC $FB23
FB20  4C 95 FB   JMP $FB95
FB23  A6 A7      LDX $A7
FB25  CA         DEX
FB26  F0 2D      BEQ $FB55
FB28  A5 93      LDA $93
FB2A  F0 0C      BEQ $FB38
FB2C  A0 00      LDY #$00
FB2E  A5 BD      LDA $BD
FB30  D1 AC      CMP ($AC),Y
FB32  F0 04      BEQ $FB38
FB34  A9 01      LDA #$01
FB36  85 B6      STA $B6
FB38  A5 B6      LDA $B6
FB3A  F0 4B      BEQ $FB87
```

```
FB3C  A2 3D     LDX #$3D
FB3E  E4 9E     CPX $9E
FB40  90 3E     BCC $FB80
FB42  A6 9E     LDX $9E
FB44  A5 AD     LDA $AD
FB46  9D 01 01  STA $0101,X
FB49  A5 AC     LDA $AC
FB4B  9D 00 01  STA $0100,X
FB4E  E8        INX
FB4F  E8        INX
FB50  86 9E     STX $9E
FB52  4C 87 FB  JMP $FB87

FB55  A6 9F     LDX $9F
FB57  E4 9E     CPX $9E
FB59  F0 35     BEQ $FB90
FB5B  A5 AC     LDA $AC
FB5D  DD 00 01  CMP $0100,X   Fehlerkorrektur bei Pass 2
FB60  D0 2E     BNE $FB90
FB62  A5 AD     LDA $AD
FB64  DD 01 01  CMP $0101,X
FB67  D0 27     BNE $FB90
FB69  E6 9F     INC $9F
FB6B  E6 9F     INC $9F
FB6D  A5 93     LDA $93
FB6F  F0 0B     BEQ $FB7C
FB71  A5 BD     LDA $BD       gelesenes Byte
FB73  A0 00     LDY #$00
FB75  D1 AC     CMP ($AC),Y   mit Speicherinhalt vergleichen
FB77  F0 17     BEQ $FB90
FB79  C8        INY
FB7A  84 B6     STY $B6
FB7C  A5 B6     LDA $B6
FB7E  F0 07     BEQ $FB87
FB80  A9 10     LDA #$10      'second pass error'
FB82  20 6A FE  JSR $FE6A     STatus setzen
FB85  D0 09     BNE $FB90
FB87  A5 93     LDA $93       Verify ?
FB89  D0 05     BNE $FB90     ja
FB8B  A8        TAY
FB8C  A5 BD     LDA $BD       gelesenes Byte
FB8E  91 AC     STA ($AC),Y   speichern
FB90  20 1B FD  JSR $FD1B     Adresszeiger erhöhen
FB93  D0 3A     BNE $FBCF
FB95  A9 80     LDA #$80
FB97  85 AA     STA $AA
FB99  A6 BE     LDX $BE       Pass-Zähler
FB9B  CA        DEX
FB9C  30 02     BMI $FBA0
FB9E  86 BE     STX $BE
FBA0  C6 A7     DEC $A7
FBA2  F0 08     BEQ $FBAC
FBA4  A5 9E     LDA $9E
FBA6  D0 27     BNE $FBCF
FBA8  85 BE     STA $BE
FBAA  F0 23     BEQ $FBCF
FBAC  20 CF FC  JSR $FCCF     ein Pass beendet
FBAF  20 D2 FB  JSR $FBD2     Adressse wieder auf Programmanfang
FBB2  A0 00     LDY #$00
FBB4  84 AB     STY $AB
```

```
FBB6  B1 AC       LDA ($AC),Y    Programm Checksumme berechnen
FBB8  45 AB       EOR $AB
FBBA  85 AB       STA $AB
FBBC  20 1B FD    JSR $FD1B      Adresszeiger erhöhen
FBBF  20 11 FD    JSR $FD11      Endadresse schon erreicht ?
FBC2  90 F2       BCC $FBB6      nein, weiter vergleichen
FBC4  A5 AB       LDA $AB        berechnete Checksumme
FBC6  45 BD       EOR $BD        mit Checksumme vom Band vergleichen
FBC8  F0 05       BEQ $FBCF      Checksumme ok ?
FBCA  A9 20       LDA #$20       'checksum error'
FBCC  20 6A FE    JSR $FE6A      STatus setzen
FBCF  4C 56 FF    JMP $FF56      Rückkehr vom Interrupt

*****************************    Adresszeiger auf Anfang setzen
FBD2  A5 C2       LDA $C2
FBD4  85 AD       STA $AD
FBD6  A5 C1       LDA $C1
FBD8  85 AC       STA $AC
FBDA  60          RTS

*****************************    Bitzähler für serielle Ausgabe setzen
FBDB  A9 08       LDA #$08
FBDD  85 A3       STA $A3
FBDF  A9 00       LDA #$00
FBE1  85 A4       STA $A4
FBE3  85 A8       STA $A8
FBE5  85 9B       STA $9B
FBE7  85 A9       STA $A9
FBE9  60          RTS
*****************************    Ein Bit auf Band schreiben
FBEA  A5 BD       LDA $BD        Bit in $BD
FBEC  4A          LSR            Bit 0 ins Carry
FBED  A9 60       LDA #$60       Zeit für '1' Bit
FBEF  90 02       BCC $FBF3
FBF1  A9 B0       LDA #$B0       Zeit für '0' Bit
FBF3  A2 00       LDX #$00
FBF5  8D 28 91    STA $9128      Timer low
FBF8  8E 29 91    STX $9129      Timer high
FBFB  AD 20 91    LDA $9120
FBFE  49 08       EOR #$08       Ausgabebit für Band invertieren
FC00  8D 20 91    STA $9120
FC03  29 08       AND #$08
FC05  60          RTS
FC06  38          SEC
FC07  66 AD       ROR $AD
FC09  30 3C       BMI $FC47

*****************************    Interrupt-Routine für Band schreiben
FC0B  A5 A8       LDA $A8
FC0D  D0 12       BNE $FC21
FC0F  A9 10       LDA #$10
FC11  A2 01       LDX #$01
FC13  20 F5 FB    JSR $FBF5      Takt auf Band schreiben
FC16  D0 2F       BNE $FC47
FC18  E6 A8       INC $A8
FC1A  A5 AD       LDA $AD
FC1C  10 29       BPL $FC47
FC1E  4C 95 FC    JMP $FC95      zweiten BLock schreiben
FC21  A5 A9       LDA $A9
FC23  D0 09       BNE $FC2E
```

```
FC25   20 F1 FB    JSR $FBF1      '0' schreiben
FC28   D0 1D       BNE $FC47
FC2A   E6 A9       INC $A9
FC2C   D0 19       BNE $FC47
FC2E   20 EA FB    JSR $FBEA      Bit auf Band schreiben
FC31   D0 14       BNE $FC47
FC33   A5 A4       LDA $A4
FC35   49 01       EOR #$01
FC37   85 A4       STA $A4
FC39   F0 0F       BEQ $FC4A
FC3B   A5 BD       LDA $BD
FC3D   49 01       EOR #$01       Bit für Ausgabe invertieren
FC3F   85 BD       STA $BD
FC41   29 01       AND #$01
FC43   45 9B       EOR $9B
FC45   85 9B       STA $9B
FC47   4C 56 FF    JMP $FF56      Rückkehr vom Interrupt

FC4A   46 BD       LSR $BD        nächstes Bit in Position 0
FC4C   C6 A3       DEC $A3        Bitzähler erniedrigen
FC4E   A5 A3       LDA $A3
FC50   F0 3A       BEQ $FC8C
FC52   10 F3       BPL $FC47      nächstes Bit ausgeben
FC54   20 DB FB    JSR $FBDB      Bitzähler wieder auf 8 setzen
FC57   58          CLI
FC58   A5 A5       LDA $A5
FC5A   F0 12       BEQ $FC6E
FC5C   A2 00       LDX #$00
FC5E   86 D7       STX $D7
FC60   C6 A5       DEC $A5
FC62   A6 BE       LDX $BE
FC64   E0 02       CPX #$02
FC66   D0 02       BNE $FC6A
FC68   09 80       ORA #$80
FC6A   85 BD       STA $BD
FC6C   D0 D9       BNE $FC47
FC6E   20 11 FD    JSR $FD11      Endadresse schon erreicht ?
FC71   90 0A       BCC $FC7D
FC73   D0 91       BNE $FC06
FC75   E6 AD       INC $AD
FC77   A5 D7       LDA $D7
FC79   85 BD       STA $BD
FC7B   B0 CA       BCS $FC47
FC7D   A0 00       LDY #$00
FC7F   B1 AC       LDA ($AC),Y    zu schriebenedes Byte
FC81   85 BD       STA $BD
FC83   45 D7       EOR $D7
FC85   85 D7       STA $D7
FC87   20 1B FD    JSR $FD1B      Adresszeiger erhöhen
FC8A   D0 BB       BNE $FC47
FC8C   A5 9B       LDA $9B
FC8E   49 01       EOR #$01
FC90   85 BD       STA $BD
FC92   4C 56 FF    JMP $FF56      Rückkehr vom Interrupt

FC95   C6 BE       DEC $BE        Zähler für Blocks erniedrigen
FC97   D0 03       BNE $FC9C      noch ein Block ?
FC99   20 08 FD    JSR $FD08      nein, Band-Motor aus
FC9C   A9 50       LDA #$50
FC9E   85 A7       STA $A7
```

```
FCA0  A2 08       LDX #$08
FCA2  78          SEI
FCA3  20 FB FC    JSR $FCFB     IRQ setzen
FCA6  D0 EA       BNE $FC92     Rückkehr vom Interrupt

******************************  Interruptroutine für Band schreiben
FCA8  A9 78       LDA #$78
FCAA  20 F3 FB    JSR $FBF3     Bit auf Band schreiben
FCAD  D0 E3       BNE $FC92
FCAF  C6 A7       DEC $A7
FCB1  D0 DF       BNE $FC92     Rückkehr vom Interrupt
FCB3  20 DB FB    JSR $FBDB     Bitzähler für serielle Ausgabe setzen
FCB6  C6 AB       DEC $AB
FCB8  10 D8       BPL $FC92
FCBA  A2 0A       LDX #$0A
FCBC  20 FB FC    JSR $FCFB     IRQ-Vektor setzen
FCBF  58          CLI
FCC0  E6 AB       INC $AB
FCC2  A5 BE       LDA $BE
FCC4  F0 30       BEQ $FCF6
FCC6  20 D2 FB    JSR $FBD2     Adresse wieder auf Anfang setzen
FCC9  A2 09       LDX #$09
FCCB  86 A5       STX $A5
FCCD  D0 85       BNE $FC54
FCCF  08          PHP
FCD0  78          SEI
FCD1  20 08 FD    JSR $FD08     Band-Motor ausschalten
FCD4  A9 7F       LDA #$7F
FCD6  8D 2E 91    STA $912E
FCD9  A9 F7       LDA #$F7
FCDB  8D 20 91    STA $9120
FCDE  A9 40       LDA #$40
FCE0  8D 2B 91    STA $912B
FCE3  20 39 FE    JSR $FE39
FCE6  AD A0 02    LDA $02A0
FCE9  F0 09       BEQ $FCF4
FCEB  8D 15 03    STA $0315     IRQ wieder auf Standard
FCEE  AD 9F 02    LDA $029F
FCF1  8D 14 03    STA $0314
FCF4  28          PLP
FCF5  60          RTS

******************************  IRQ-Vektor aus Tabelle setzen
FCF6  20 CF FC    JSR $FCCF     IRQ auf Standard
FCF9  F0 97       BEQ $FC92
FCFB  BD E9 FD    LDA $FDE9,X
FCFE  8D 14 03    STA $0314     IRQ low
FD01  BD EA FD    LDA $FDEA,X
FD04  8D 15 03    STA $0315     IRQ high
FD07  60          RTS

******************************  Band-Motor ausschalten
FD08  AD 1C 91    LDA $911C
FD0B  09 0E       ORA #$0E
FD0D  8D 1C 91    STA $911C
FD10  60          RTS

******************************  prüft auf Erreichen der Endadresse
FD11  38          SEC
FD12  A5 AC       LDA $AC       laufende Adresse in $AC/$AD
```

```
FD14  E5 AE       SBC $AE
FD16  A5 AD       LDA $AD        Endadresse in $AE/$AF
FD18  E5 AF       SBC $AF
FD1A  60          RTS

*****************************    Zeiger erhöhen
FD1B  E6 AC       INC $AC
FD1D  D0 02       BNE $FD21      $AC/$AD
FD1F  E6 AD       INC $AD
FD21  60          RTS

*****************************    RESET-Routine
FD22  A2 FF       LDX #$FF
FD24  78          SEI            Stackpointer initialisieren
FD25  9A          TXS
FD26  D8          CLD
FD27  20 3F FD    JSR $FD3F      ROM in $A000 ?
FD2A  D0 03       BNE $FD2F
FD2C  6C 00 A0    JMP ($A000)    Sprung in ROM-Modul-Programm

FD2F  20 8D FD    JSR $FD8D      Pointer setzen, RAM-Bereich feststellen
FD32  20 52 FD    JSR $FD52      I/O-Vektoren $0314-$0335 setzen
FD35  20 F9 FD    JSR $FDF9      I/O-Register initialisieren
FD38  20 18 E5    JSR $E518      Videocontroller und CLR SCREEN
FD3B  58          CLI
FD3C  6C 00 C0    JMP ($C000)    JMP $E378  Sprung zum BASIC-Kaltstart

*****************************    prüft auf ROM in $A000
FD3F  A2 05       LDX #$05
FD41  BD 4C FD    LDA $FD4C,X
FD44  DD 03 A0    CMP $A003,X    vergleicht mit Identifikation
FD47  D0 03       BNE $FD4C
FD49  CA          DEX
FD4A  D0 F5       BNE $FD41
FD4C  60          RTS

*****************************    ROM-Modul Identifikation
FD4D  41 30 C3 C2 CD             'a0CBM'

*****************************    setzt I/O-Vektoren
FD51  A2 6D       LDX #$6D
FD54  A0 FD       LDY #$FD       Zeiger auf Vektorentabelle
FD56  18          CLC

*****************************    holt (C=1)/ setzt (C=0) I/O-Vektoren
FD57  86 C3       STX $C3
FD59  84 C4       STY $C4        Zeiger auf Tabelle in X/Y
FD5B  A0 1F       LDY #$1F
FD5D  B9 14 03    LDA $0314,Y
FD60  B0 02       BCS $FD64      Vektoren setzen/holen
FD62  B1 C3       LDA ($C3),Y
FD64  91 C3       STA ($C3),Y
FD66  99 14 03    STA $0314,Y
FD69  88          DEY
FD6A  10 F1       BPL $FD5D
FD6C  60          RTS

*****************************    Tabelle der Hardware- und I/O-Vektoren
FD6D  BF EA D2 FE AD FE 0A F4
FD75  4A F3 C7 F2 09 F3 F3 F3
```

```
FD7D  0E F2 7A F2 70 F7 F5 F1
FD85  EF F3 D2 FE 49 F5 85 F6

*****************************   Arbeitsspeicher initialisieren
FD8D  A9 00       LDA #$00
FD8F  AA          TAX
FD90  95 00       STA $00,X
FD92  9D 00 02    STA $0200,X   Zeropage und Page 2+3 löschen
FD95  9D 00 03    STA $0300,X
FD98  E8          INX
FD99  D0 F5       BNE $FD90
FD9B  A2 3C       LDX #$3C
FD9D  A0 03       LDY #$03
FD9F  86 B2       STX $B2       Bandpufferzeiger auf $033C
FDA1  84 B3       STY $B3
FDA3  85 C1       STA $C1
FDA5  85 97       STA $97
FDA7  8D 81 02    STA $0281
FDAA  A8          TAY
FDAB  A9 04       LDA #$04
FDAD  85 C2       STA $C2
FDAF  E6 C1       INC $C1
FDB1  D0 02       BNE $FDB5
FDB3  E6 C2       INC $C2
FDB5  20 91 FE    JSR $FE91     findet RAM-Start für BASIC
FDB8  A5 97       LDA $97
FDBA  F0 22       BEQ $FDDE
FDBC  B0 F1       BCS $FDAF
FDBE  A4 C2       LDY $C2
FDC0  A6 C1       LDX $C1
FDC2  C0 20       CPY #$20
FDC4  90 25       BCC $FDEB
FDC6  C0 21       CPY #$21
FDC8  B0 08       BCS $FDD2
FDCA  A0 1E       LDY #$1E      und RAM-Ende
FDCC  8C 88 02    STY $0288
FDCF  4C 7B FE    JMP $FE7B

FDD2  A9 12       LDA #$12
FDD4  8D 82 02    STA $0282     BASIC-Start auf $1200
FDD7  A9 10       LDA #$10
FDD9  8D 88 02    STA $0288     Video-RAM auf $1000
FDDC  D0 F1       BNE $FDCF
FDDE  90 CF       BCC $FDAF
FDE0  A5 C2       LDA $C2
FDE2  8D 82 02    STA $0282
FDE5  85 97       STA $97
FDE7  C9 11       CMP #$11
FDE9  90 C4       BCC $FDAF
FDEB  20 C3 E5    JSR $E5C3     Videocontroller initialisieren
FDEE  4C EB FD    JMP $FDEB

*****************************   Tabelle der IRQ-Vektoren
FDF1  A8 FC 0B FC BF EA 8E F9

*****************************   I/O-Register initialisieren
FDF9  A9 7F       LDA #$7F
FDFB  8D 1E 91    STA $911E
FDFE  8D 2E 91    STA $912E
FE01  A9 40       LDA #$40
```

```
FE03  8D 2B 91    STA $912B
FE06  A9 40       LDA #$40
FE08  8D 1B 91    STA $911B
FE0B  A9 FE       LDA #$FE
FE0D  8D 1C 91    STA $911C
FE10  A9 DE       LDA #$DE
FE12  8D 2C 91    STA $912C
FE15  A2 00       LDX #$00
FE17  8E 12 91    STX $9112
FE1A  A2 FF       LDX #$FF
FE1C  8E 22 91    STX $9122
FE1F  A2 00       LDX #$00
FE21  8E 23 91    STX $9123
FE24  A2 80       LDX #$80
FE26  8E 13 91    STX $9113
FE29  A2 00       LDX #$00
FE2B  8E 1F 91    STX $911F
FE2E  20 84 EF    JSR $EF84
FE31  A9 82       LDA #$82
FE33  8D 1E 91    STA $911E
FE36  20 8D EF    JSR $EF8D
FE39  A9 C0       LDA #$C0
FE3B  8D 2E 91    STA $912E
FE3E  A9 26       LDA #$26
FE40  8D 24 91    STA $9124
FE43  A9 48       LDA #$48
FE45  8D 25 91    STA $9125
FE48  60          RTS

******************************  Filenamenparameter setzen
FE49  85 B7       STA $B7       Länge
FE4B  86 BB       STX $BB       Adresse low
FE4D  84 BC       STY $BC       Adresse high
FE4F  60          RTS

******************************  Fileparameter setzen
FE50  85 B8       STA $B8       Filenummer
FE52  86 BA       STX $BA       Primäradresse
FE54  84 B9       STY $B9       Sekundäadresse
FE56  60          RTS

******************************  Status holen
FE57  A5 BA       LDA $BA       Primäradresse
FE59  C9 02       CMP #$02      RS 232 ?
FE5B  D0 0B       BNE $FE68     nein
FE5D  AD 97 02    LDA $0297     RS 232 Status (Fehler, wird wieder gelöscht!)
FE60  A9 00       LDA #$00
FE62  8D 97 02    STA $0297     Status löschen
FE65  60          RTS

******************************  Prg/Direkt-Flag setzen
FE66  85 9D       STA $9D       Flag

******************************  Status holen
FE68  A5 90       LDA $90       Status

******************************  Status setzen
FE6A  05 90       ORA $90
FE6C  85 90       STA $90
FE6E  60          RTS
```

```
******************************   TIME-OUT-Flag für IEC-Bus setzen
FE6F  8D 85 02    STA $0285
FE72  60          RTS

******************************   MEMTOP holen (C=0)/ setzen (C=1)
FE73  90 06       BCC $FE7B
FE75  AE 83 02    LDX $0283
FE78  AC 84 02    LDY $0284
FE7B  8E 83 02    STX $0283
FE7E  8C 84 02    STY $0284
FE81  60          RTS

******************************   MEMBOT holen (C=0)/ setzen (C=1)
FE82  90 06       BCC $FE8A
FE84  AE 81 02    LDX $0281
FE87  AC 82 02    LDY $0282
FE8A  8E 81 02    STX $0281
FE8D  8C 82 02    STY $0282
FE90  60          RTS

******************************   RAM-Test
FE91  B1 C1       LDA ($C1),Y    Wert merken
FE93  AA          TAX
FE94  A9 55       LDA #$55       %01010101
FE96  91 C1       STA ($C1),Y
FE98  D1 C1       CMP ($C1),Y
FE9A  D0 08       BNE $FEA4
FE9C  6A          ROR            %10101010
FE9D  91 C1       STA ($C1),Y
FE9F  D1 C1       CMP ($C1),Y
FEA1  D0 01       BNE $FEA4
FEA3  A9          .BYTE $A9
FEA4  18          CLC            C=0, dann Fehler
FEA5  8A          TXA
FEA6  91 C1       STA ($C1),Y    alten Wert wieder zurückschreiben
FEA8  60          RTS

******************************   NMI-Routine
FEA9  78          SEI
FEAA  6C 18 03    JMP ($0318)    JMP $FEAD
FEAD  48          PHA
FEAE  8A          TXA
FEAF  48          PHA
FEB0  98          TYA
FEB1  48          PHA
FEB2  AD 1D 91    LDA $911D      kein Interrupt (nur RESTORE-Taste) ?
FEB5  10 48       BPL $FEFF      Rückkehr vom Interrupt
FEB7  2D 1E 91    AND $911E
FEBA  AA          TAX
FEBB  29 02       AND #$02       RS 232 aktiv ?
FEBD  F0 1F       BEQ $FEDE      ja
FEBF  20 3F FD    JSR $FD3F      ROM in $A000 ?
FEC2  D0 03       BNE $FEC7
FEC4  6C 02 A0    JMP ($A002)    ja, dann Sprung auf Modul-NMI

FEC7  2C 11 91    BIT $9111      Flags rücksetzen
FECA  20 34 F7    JSR $F734      Uhrzeit erhöhen und STOP-Taste abfragen
FECD  20 E1 FF    JSR $FFE1      STOP-Taste gedrückt ?
FED0  D0 2D       BNE $FEFF      nein, dann RTI
FED2  20 52 FD    JSR $FD52      Vektoren $0314-$0335 setzen
```

```
FED5  20 F9 FD    JSR $FDF9      I/O-Register initialisieren
FED8  20 18 E5    JSR $E518      Videocontroller und CLR SCREEN
FEDB  6C 02 C0    JMP ($C002)    JMP $E467  zum BASIC-Warmstart

*****************************    NMI Routine für RS 232
FEDE  AD 1E 91    LDA $911E      Interrupt Enable Register
FEE1  09 80       ORA #$80       Bit 7 für Daten Schreiben
FEE3  48          PHA            merken
FEE4  A9 7F       LDA #$7F
FEE6  8D 1E 91    STA $911E      Interrupt löschen
FEE9  8A          TXA
FEEA  29 40       AND #$40
FEEC  F0 14       BEQ $FF02
FEEE  A9 CE       LDA #$CE
FEF0  05 B5       ORA $B5
FEF2  8D 1C 91    STA $911C
FEF5  AD 14 91    LDA $9114
FEF8  68          PLA
FEF9  8D 1E 91    STA $911E
FEFC  20 A3 EF    JSR $EFA3
FEFF  4C 56 FF    JMP $FF56      Rückkehr vom Interrupt

FF02  8A          TXA
FF03  29 20       AND #$20
FF05  F0 25       BEQ $FF2C
FF07  AD 10 91    LDA $9110
FF0A  29 01       AND #$01
FF0C  85 A7       STA $A7
FF0E  AD 18 91    LDA $9118
FF11  E9 16       SBC #$16
FF13  6D 99 02    ADC $0299
FF16  8D 18 91    STA $9118
FF19  AD 19 91    LDA $9119
FF1C  6D 9A 02    ADC $029A
FF1F  8D 19 91    STA $9119
FF22  68          PLA
FF23  8D 1E 91    STA $911E
FF26  20 36 F0    JSR $F036
FF29  4C 56 FF    JMP $FF56      Rückkehr vom Interrupt

FF2C  8A          TXA
FF2D  29 10       AND #$10
FF2F  F0 25       BEQ $FF56      Rückkehr vom Interrupt

FF31  AD 93 02    LDA $0293
FF34  29 0F       AND #$0F
FF36  D0 00       BNE $FF38
FF38  0A          ASL
FF39  AA          TAX
FF3A  BD 5A FF    LDA $FF5A,X
FF3D  8D 18 91    STA $9118      Timer für RS 232 Baud-Rate setzen
FF40  BD 5B FF    LDA $FF5B,X
FF43  8D 19 91    STA $9119
FF46  AD 10 91    LDA $9110
FF49  68          PLA
FF4A  09 20       ORA #$20
FF4C  29 EF       AND #$EF
FF4E  8D 1E 91    STA $911E
FF51  AE 98 02    LDX $0298
FF54  86 A8       STX $A8
```

```
FF56  68          PLA
FF57  A8          TAY
FF58  68          PLA
FF59  AA          TAX
FF5A  68          PLA
FF5B  40          RTI

*****************************   Timer-Konstanten für RS 232 Baud-Rate
FF5C  E6 2A       $2AE6 =          50 Baud
FF5E  78 1C       $1C78 =          75 Baud
FF60  49 13       $1349 =         110 Baud
FF62  B1 0F       $0FB1 =         134.5 Baud
FF64  0A 0E       $0E0A =         150 Baud
FF66  D3 06       $06D3 =         300 Baud
FF68  38 03       $0338 =         600 Baud
FF6A  6A 01       $016A =        1200 Baud
FF6C  D0 00       $00D0 =        1800 Baud
FF6E  83 00       $0083 =        2400 Baud
FF70  36 00       $0036 =        3600 Baud

*****************************   Interrupt-Handling
FF72  48          PHA
FF73  8A          TXA
FF74  48          PHA
FF75  98          TYA
FF76  48          PHA
FF77  BA          TSX
FF78  BD 04 01    LDA $0104,X    Processor-Status holen
FF7B  29 10       AND #$10
FF7D  F0 03       BEQ $FF82      BREAK-Flag testen
FF7F  6C 16 03    JMP ($0316)    BRK  $FED2
FF82  6C 14 03    JMP ($0314)    IRQ  $EABF

FF85  FF FF FF FF FF

*****************************   Sprungtabelle für Betriebssystem-Routinen

FF8A  4C 52 FD    JMP $FD52      I/O-Vektoren $0314-$0335 initialisieren

FF8D  4C 57 FD    JMP $FD57      I/O-Vektoren $0314-$0335 setzen

FF90  4C 66 FE    JMP $FE66      Direkt-Flag setzen und Status holen

FF93  4C C0 EE    JMP $EEC0      Sekundär-Adresse nach LISTEN senden

FF96  4C CE EE    JMP $EECE      Sekundär-Adresse nach TALK senden

FF99  4C 73 FE    JMP $FE73      RAM-Ende setzen/holen C=1/0

FF9C  4C 82 FE    JMP $FE82      RAM-Anfang setzen/holen C=1/0

FF9F  4C 1E EB    JMP $EB1E      Tastatur-Dekodierung

FFA2  4C 6F FE    JMP $FE6F      TIMEOUT-Flag für IEC-Bus setzen

FFA5  4C 19 EF    JMP $EF19      INPUT vom IEC-Bus

FFA8  4C E4 EE    JMP $EEE4      OUTPUT auf IEC-Bus

FFAB  4C F6 EE    JMP $EEF6      UNTALK senden
```

```
FFAE  4C 04 EF    JMP $EF04      UNLISTEN senden

FFB1  4C 17 EE    JMP $EE17      LISTEN senden

FFB4  4C 14 EE    JMP $EE14      TALK senden

FFB7  4C 57 FE    JMP $FE57      STatus holen

FFBA  4C 50 FE    JMP $FE50      setzt Filenr(A) Prim.(X) und Sekadr.(Y)

FFBD  4C 49 FE    JMP $FE49      setzt Länge(A) + Adr.(X/Y) Filename

FFC0  6C 1A 03    JMP ($031A)    OPEN  $F40A

FFC3  6C 1C 03    JMP ($031C)    CLOSE  $F34A

FFC6  6C 1E 03    JMP ($031E)    CHKIN  $F2C7   Inputgerät festlegen

FFC9  6C 20 03    JMP ($0320)    CKOUT  $F309   Ausgabegerät festlegen

FFCC  6C 22 03    JMP ($0322)    CLRCH  $F3F3   Rücksetzen des I/O-Kanals

FFCF  6C 24 03    JMP ($0324)    BASIN  $F20E   Eingabe eines Zeichens

FFD2  6C 26 03    JMP ($0326)    BSOUT  $F27A   Ausgabe eines Zeichens

FFD5  4C 42 F5    JMP $F542      LOAD

FFD8  4C 75 F6    JMP $F675      SAVE

FFDB  4C 67 F7    JMP $F767      TIME setzen

FFDE  4C 60 F7    JMP $F760      TIME holen

FFE1  6C 28 03    JMP ($0328)    STOP-Taste abfragen  $F770

FFE4  6C 2A 03    JMP ($032A)    GETIN  $F1F5  ein Zeichen holen

FFE7  6C 2C 03    JMP ($032C)    CLALL  $F3EF  I/O-Kanäle schließen

FFEA  4C 34 F7    JMP $F734      UDTIM  Uhr und Stoptaste

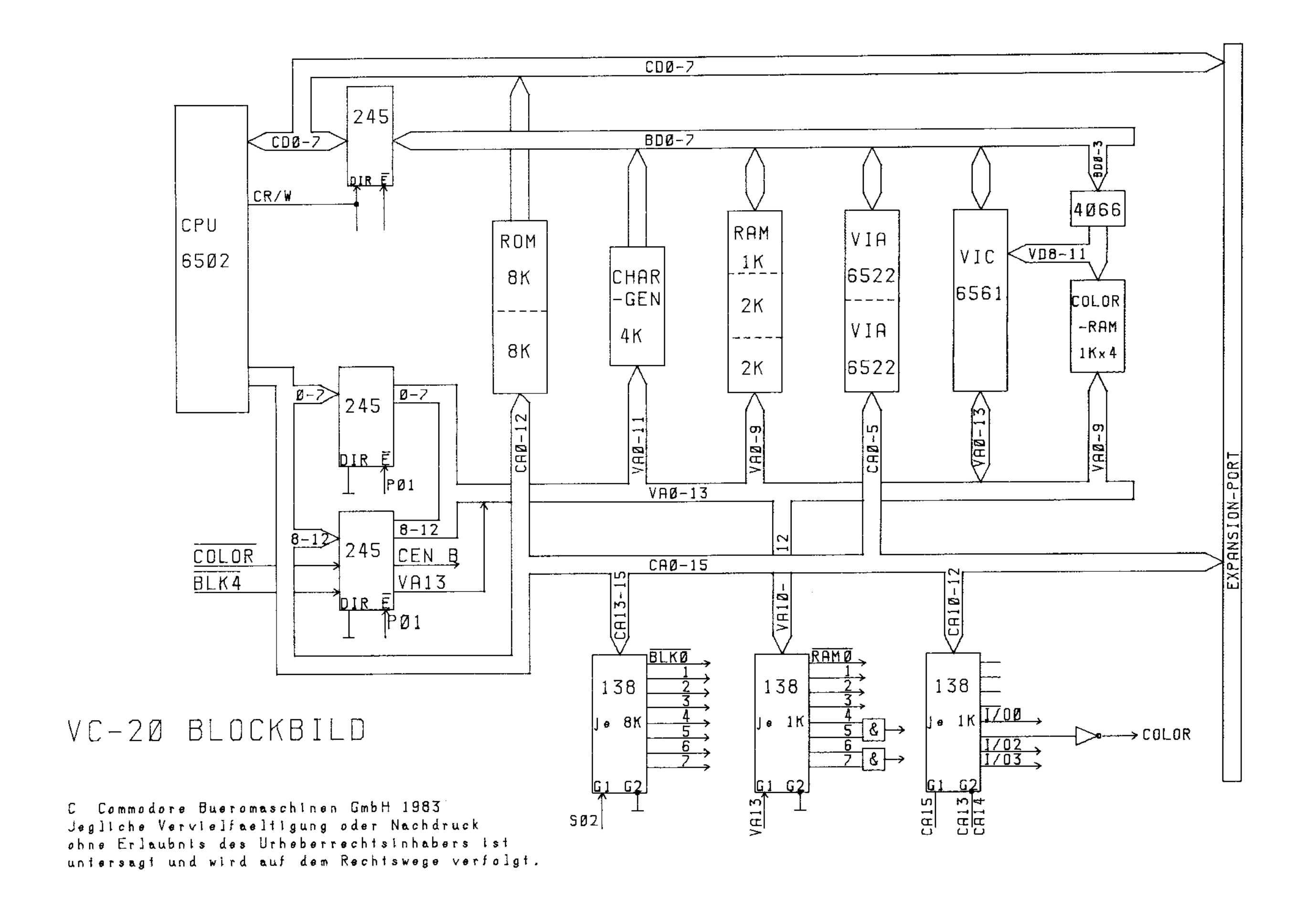
FFED  4C 05 E5    JMP $E505      holt Bildschirmgröße nach X/Y

FFF0  4C 0A E5    JMP $E50A      holt/setzt Cursorposition (X/Y)

FFF3  4C 00 E5    JMP $E500      holt VIA 6522 Adresse ($9110) nach X/Y

FFF6  FF FF FF FF

FFFA  A9 FE       $FEA9          NMI-Vektor
FFFC  22 FD       $FD22          RESET-Vektor
FFFE  72 FF       $FF72          IRQ-Vektor
```



EXPANSION-PORT
CD0-7
BD0-7
245
DIR E
CR/W
CPU
6502
ROM
8K
8K
CA0-12
CHAR
-GEN
4K
VA0-11
RAM
1K
2K
2K
VA0-9
VIA
6522
VIA
6522
CA0-5
VIC
6561
VA0-13
VD8-11
4066
BD0-3
COLOR
-RAM
1Kx4
VA0-9
0-7
245
0-7
DIR E
P01
VA0-13
12
8-12
245
8-12
CEN B
VA13
DIR E
P01
COLOR
BLK4
CA0-15
CA13-15
138
Je 8K
G1 G2
S02
BLK0
1
2
3
4
5
6
7
VA10-
138
Je 1K
G1 G2
VA13
RAM0
1
2
3
4
5
6
7
&
&
CA10-12
138
Je 1K
G1 G2
CA15
CA13
CA14
I/00
I/02
I/03
COLOR
VC-20 BLOCKBILD
C Commodore Bueromaschinen GmbH 1983
Jegliche Vervielfaeltigung oder Nachdruck
ohne Erlaubnis des Urheberrechtsinhabers ist
untersagt und wird auf dem Rechtswege verfolgt.

Einleitung

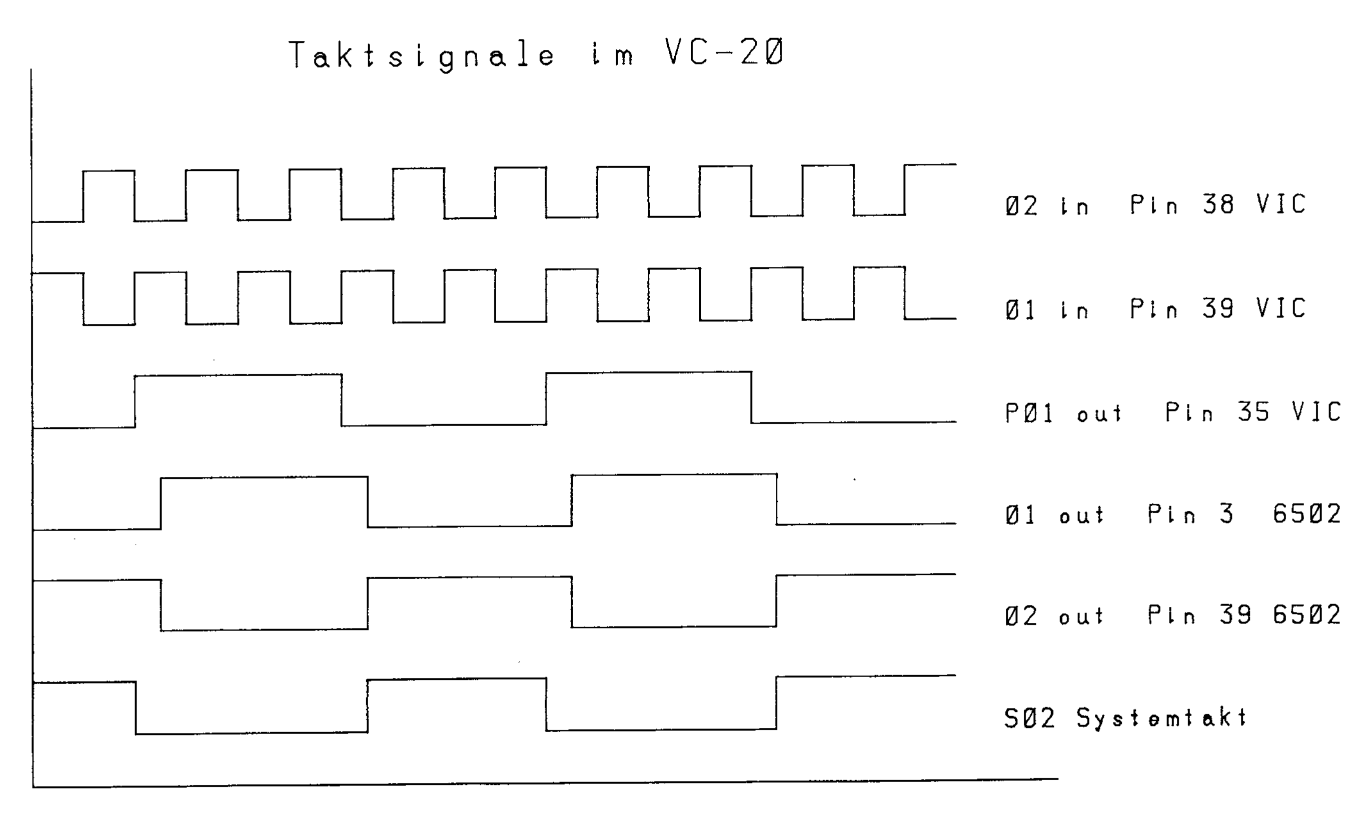
Der VC-20 ist ein phantastischer Computer.
Wenn man einmal etwas zurückschaut, dann ist es noch nicht lange her, da mußte man für den legendären KIM 1, den ersten Computer der Firma Commodore, gut 800 DM bezahlen. Dieser KIM 1 hatte kein Gehäuse, verfügte über eine kleine Hexadezimal-Tastatur und für die Ausgabe waren sechs LED-Anzeigen eingebaut. Damit konnte man erste Gehversuche in Maschinensprache machen.
Heute bekommen Sie für knapp die Hälfte ein Gerät, das mit einer vollständigen Tastatur und einem hervorragenden Basic-Interpreter, mit Anschluß an Farbfernseher und Möglichkeit der einfachsten Erweiterung ausgestattet ist.
Diese ganzen Möglichkeiten und der niedrige Preis haben sicherlich maßgeblich dafür gesorgt, daß der VC-20 in geradezu phantastischen Stückzahlen verkauft wurde. Entsprechend groß ist mittlerweile auch das Angebot an Software für den VC-20.
Allerdings war es immer sehr schwierig, Schaltbilder des VC-20 zu erhalten. Das war umso ärgerlicher, als damit der Weg zu selbstgebauten Steuerungen verschlossen war.
Dem können wir in dieser Ausgabe von VC-20 INTERN abhelfen. In diesem Buch finden Sie einen kompletten Schaltplan und ein Blockschaltbild des VC-20. Den Schaltplan haben wir ausführlich dokumentiert. Wenn Sie einige Vorkenntnisse der Digitaltechnik haben, sollte Ihnen das Verstehen der Hardware des VC-20 leichtfallen.
Aber auch wenn Sie nicht vorhaben, Ihren VC-20 durch Selbstbau zu erweitern, ist eine detaillierte Kenntnis der Hardware nützlich, da dadurch die internen Zusammenhänge klarer werden.
Für Hardware- und Software-'Freaks' gleichermaßen interessant ist sicher die detailierte Registerbeschreibung der Peripherie-Bausteine im VC-20, die wir mit in dies Buch aufgenommen haben.
Zum Abschluß noch eine kurze Bemerkung. Leider hat die Firma Commodore im Laufe der Zeit verschiedene Leiterplatten im VC-20 verwendet. Der von Commodore zur Verfügung gestellte und von uns abgedruckte Schaltplan stellt den aktuellen Stand dar. Das bedeutet, daß Ihr VC-20 möglicherweise einen etwas anderen Aufbau haben kann. Diese Unterschiede sind aber relativ geringfügig. In der Hauptsache ist in den älteren Geräten der RAM-Bereich ganz mit den ICs 2114 aufgebaut. Sicheres äußeres Unterscheidungsmerkmal ist die Stromversorgung. Wenn Ihr VC-20 ein schwarzes Trafogehäuse hat, müssen Sie leider mit diesen kleinen hardwaremäßigen Unterschieden rechnen. Die Belegung der verschiedenen Ports ist aber an allen VC-20 gleich.

Die Stromversorgung.

Die Stromversorgung des VC-20 ist recht einfach und übersichtlich aufgebaut.
über die Buchse P7 ist der VC-20 mit dem Stromversorgungsgehäuse verbunden. In diesem Gehäuse befindet sich der Netztrafo und eine Gleichrichter- und Stabilisationsschaltung.
Der Anschluß an den VC-20 wird mit einer 7-poligen DIN-Steckverbindung vorgenommen.
Auf den Kontakten 4 und 5 dieser Steckverbindung liegt die im Stromversorgungsgehäuse bereits gleichgerichtete und stabilisierte 5 Volt Gleichspannung. Diese Spannung versorgt die gesamte Elektronik des VC 20. Die Kondensatoren C31, C32, C33, C34 und C40 sowie die Spule L3 filtern eventuelle Störimpulse aus, die Leuchtdiode CN1 (kurz LED) ist die Einschaltkontrolle.
Die Kontakte 1, 2 und 3 sind der Massekontakt für die 5 Volt. An den Kontakten 6 und 7 liegt eine Wechselspannung von ca. 9 Volt. diese Spannung ist mit der Sicherung F1 abgesichert. Diese Sicherung hat einen Wert von 1 Ampere. Hinter der Sicherung befindet sich ein Filter mit den Kondensatoren C35 und C36 und der Spule L4. Das Filter beseitigt Störungen, die über den Transformator aus dem Lichtnetz in den VC-20 gelangt sind.
Die Wechselspannung wird über zwei Drähte zum User-Port P1 geführt und stehen Ihnen an den Kontakten 10 und 11 zur Verfügung. An diesen Kontakten können Sie die Wechselspannung abgreifen und nach entsprechender Gleichrichtung und Stabilisierung für Ihre eignenen Erweiterungen verwenden.
Parallel zu den Kontakten am User-Port liegt die Wechselspannung noch an dem Brückengleichrichter CR 2. Die vier Dioden im CR2 erzeugen aus der Wechselspannung eine pulsierende Gleichspannung, die mit dem Kondensator C39 geglättet wird. über dem Kondensator baut sich eine unstabilisierte Gleichspannung von 9 Volt auf. Benutzt wird die Gleichspannung zum Betrieb des Motors in der Datasette. Auf die Datasette und ihren Anschluß werden wir weiter unten noch genauer eingehen.
Wenn Sie die Datasette nicht benutzen, können Sie die Wechselspannung mit ca. 1 Ampere belasten. Damit lassen sich schon ganz schön umfangreiche Zusatzschaltungen versorgen. Wenn Sie die Datasette aber doch benutzen, so sollten Sie die Wechselspannung nur mit maximal 100 mA belasten. Bei höherer Belastung wird die Sicherung F1 ihren 'Geist' aufgeben.
Im Gegensatz zur Wechselspannung ist die vom Stromversorgungsgehäuse gelieferte 5 Volt Gleichspannung kurzschlußfest. Diese Spannung können Sie am User-Port-Kontakt 2, am Expansion-Slot-Kontakt 21, am Controller-Port Pin 7 oder aber am Video/Audio-Port Pin 1 abgreifen. Auch hier sollten Sie den Maximalstrom von 100 mA nicht überschreiten.

Die Takterzeugung.

Der Takt in einem Computersystem ist vergleichbar mit dem Pendel einer Uhr. Ohne Pendel-Takt steht die Uhr, je genauer der Takt, desto genauer geht die Uhr. Entsprechend verhält sich der Takt im VC-20. Ohne Takt geht also nichts.
Um den Takt möglichst gleichmäßig schlagen zu lassen, benutzt man in der Regel einen Schwingquarz. Ein solcher Quarz liefert eine sehr exakte Frequenz, die sich nur ganz geringfügig bei Temparatur- und Spannungsschwankungen ändert. Das ist im Fall des VC-20 sehr wichtig, da die Quarzfrequenz bei der Erzeugung der Signale für das Fernsehbild eine große Rolle spielt.
Realisiert ist der Taktoszillator mit dem Quarz Y1 und dem IC UB9.
Der Quarz ist schon bei der Produktion auf eine Frequenz von 4.433618 MHz abgeglichen.
Das IC UB9 ist ein TTL-IC 7402 mit vier NOR-Gattern. Ein NOR-Gatter stellt eine logische Oder-Verknüpfung dar, deren Ausgang invertiert ist. Alle vier NOR-Gatter im UB9 sind allerdings ausschließlich als Inverter beschaltet, die beiden Eingänge eines jeden Gatters sind zusammengeschaltet.
Die beiden Gatter an den Pins 1, 2, 3, 4, 5 und 6 bilden zusammen mit dem Quarz, den Widerständen R5 und R6 und den Kondensatoren C50 und C48 die Oszillatorschaltung. Der Kondensator C48 ist ein Trimmkondensator, mit seiner Hilfe läßt sich die Frequenz geringfügig einstellen. An den Ausgängen Pin 1 und Pin 4 wird das Ausgangssignal abgenommen. Dabei sind die beiden Signale um 180 Grad phasenverschoben, d.h. Wenn der Pin 4 einen High-Pegel hat, liegt Pin 1 auf Low und umgekehrt. Die beiden im UB9 noch freien Gatter 'verstärken' die Taktsignale, die Schaltflanken zwischen den logischen Low- und Highzuständen werden steiler. Diese Signale werden an die Pins 38 und 39 des Video-Interface-Chips geführt.
Auf den Video-Interface-Chip 6561 (kurz VIC genannt) werden wir später detailiert eingehen. Zur Takterzeugung interresiert uns im Moment nur die Erzeugung des Prozessortaktsignals.
Der 6561 enthält eine Frequenzteilerstufe, welche die Eingangsfrequenz durch vier teilt. Damit entsteht eine Frequenz von 1.1084 MHz. Diese neue Frequenz steht an Pin 35 des VIC zur Verfügung und wird an den Eingang Pin 11 des Inverters UB4 geführt. Vom Ausgang Pin 10 des Inverters wird das Signal auf den Takteingang des Prozessors Pin 3 geführt.
Der Prozessor 6502 erzeugt aus dem Signal 00 die beiden Taktsignale 01 Out und 02 Out. Diese beiden Signale sind zueinander um 180 Grad phasenverschoben. Das Signal 02 hat die gleiche Phasenlage wie das Eingangsignal 00, es ist nur zeitlich ein wenig verzögert.
Zur besseren übersicht sind die gesamten Taktsignale als Timingdiagramm auf der nächsten Seite dargestellt.


Taktsignale im VC-2Ø
Ø2 in Pin 38 VIC
Ø1 in Pin 39 VIC
PØ1 out Pin 35 VIC
Ø1 out Pin 3 65Ø2
Ø2 out Pin 39 65Ø2
SØ2 Systemtakt

Wichtig innerhalb des Timings ist das Signal 02. Es stellt das Bezugssignal für alle Operationen des Prozessors dar. Speicherzugriffe des Prozessors finden nur in den High-Phasen des 02 statt, in den Low-Phasen werden die nächsten Lese- oder Schreiboperationen vorbereitet, der Bus ist frei. Diese Lücken werden geschickt vom VIC ausgenutzt, um auf das Videoram und den Charakter-ROM zuzugreifen.

Der Prozessor 6502

Der im VC 20 eingesetzte Prozessor ist der bekannte 6502. Dieser Prozessor ist seit den Tagen des legendären PET 2001 in jedem Computer der Firma Commodore eingebaut worden.
Der 6502 ist einer der leistungsfähigsten 8-Bit-Prozessoren. Die Bezeichnung 8-Bit-Prozessor bezieht sich auf den Datenbus. Dieser Datenbus ist 8 Bit breit. Damit sind 256 verschiedene Zustände auf den Datenleitungen möglich. Der Adressbus ist 16 Bit breit. Mit diesen 16 Adressleitungen kann der Prozessor 65536 verschiedene Speicherzellen adressieren. Anders ausgedrückt kann der Prozessor 64 k Speicher adressieren.
Aber schauen wir uns die einzelnen Anschlüsse des 6502 einmal der Reihe nach an.

Pin	Bez.	Funktion
1	Vss	Betriebsspannung Masse
2	RDY	Eingang, im VC-20 nicht benutzt
3	01out	Ausgang, Takt 01
4	-IRQ	Eingang, Interrupt ReQuest
5	N.C.	keine Funktion
6	-NMI	Eingang, Non-Maskable Interrupt
7	SYNC	Ausgang, im VC-20 nicht benutzt
8	Vcc	Betriebsspannung +5V
9	CA0	Ausgang, Adressbit 0
bis		
20	CA11	Ausgang, Adressbit 11
21	Vss	Betriebsspannung Masse
22	CA12	Ausgang, Adressbit 12
bis		
25	CA15	Ausgang, Adressbit 15
26	CD7	Datenbus, bidirektional Bit 7
bis		
33	CD0	Datenbus, bidirektional Bit 0
34	R/-W	Ausgang, Read/-Write
35	N.C.	keine Funktion
36	N.C.	keine Funktion
37	00in	Eingang, Taktsignal 00
38	S.O.	Eingang, nicht benutzt
39	02out	Ausgang, Taktsignal 02
40	-RES	Eingang, -Reset

Der Adressbus des Prozessors

Über diese 16 Leitungen werden alle Speicherplätze angesprochen. Dabei ist zu bedenken, daß auch die Peripheriebausteine Memory Mapped, also als Speicher angesprochen werden.
Wenn Sie den Blockschaltplan auf Seite 175 genauer betrachten, werden Sie feststellen, daß der Adressbus im VC-20 gleich zwei mal vorhanden ist. Da ist zum einen der direkte Prozessor-Adressbus CA0 bis CA15. Dieser Bus liegt, wie man im Blockschaltbild sehen kann, an den 16 K Basic- und Betriebssystem-ROM, an den VIAs 6522 UAB1 und UAB3, an den Dekodern UC5 und UC6 und am Expansion Port. Zu guter letzt liegen die Adressbits 0 bis 12 an den beiden Busbuffern UD8 und UE8.
Diese Busbuffer vom Typ 74LS245 sind grundsätzlich bidirektional. Das bedeutet, daß der Datentransfer in beide Richtungen vorgenommen werden kann. Dabei wird die Datenrichtung durch das Signal DIR an Pin 1 bestimmt. Bei den Adressbusbuffern ist dieser Anschluß auf Masse gelegt, die Datenrichtung ist festgelegt.
An den so festgelegten Ausgängen der Buffer liegt der Adressbus VA0 bis VA13. Dieser Bus verbindet den Prozessor mit dem RAM und dem Farb-RAM, dem Charakter-ROM, dem VIC und dem Dekoder UC4.
Der Anschluß -ENABLE Pin 19 dieser Buffer hat eine besondere Funktion. Wenn dieser Anschluß High ist, gehen die Ausgänge unabhängig von der Datenrichtung in den Tri-State-Zustand, und der Prozessor kann die am Bus VA liegenden ICs nicht adressieren.
Wie schon erwähnt, benutzt der Prozessor den Adress- und Datenbus nur in den Zeiten, in denen 02 High ist. Das bedeutet aber nicht, das in den Low-Phasen von 02 der Prozessor seine Daten und Adressleitungen in den Tri-State-Zustand versetzt. In diesen Zeiten werden die nächsten Operationen des Prozessors vorbereitet, auf den Adressbus wird die nächste Adresse gelegt. Da aber in den 02-Low-Zeiten der VIC den Adressbus belegt, muß der Prozessoradressbus für diese Zeit abgeschaltet werden. Auf Grund dieser festen Zeiteinteilung kann der Pin 19 der Buffer direkt vom Signal P01 vom VIC gesteuert werden. Dieses Signal ist immer dann High, wenn 02 Low ist.

Der Datenbus

Über diese 8 Leitungen findet der gesamte Datentransfer zwischen Prozessor und Peripheriebausteinen statt. Auch der Datenbus im VC-20 ist, wie im Blockschaltbild gut zu sehen, doppelt vorhanden, einmal als Datenbus CD0 bis CD7, einmal als BD0 bis BD7.
Der CD-Bus ist der direkte Datenbus des Prozessors. Er ist mit den Basic- und Kernal-ROMs, dem Busbuffer UF8 und dem Expansion Port verbunden.
Der BD-Bus ist der gepufferte Datenbus hinter dem IC UF8. Bei diesem Datenbusbuffer wird von der Möglichkeit der Datenrichtungsumschaltung Gebrauch gemacht. Der entsprechende

Eingang Pin 1 ist verbunden mit der R/-W-Leitung des Prozessors. Wenn der Prozessor Daten liest, ist dieser Prozessorpin High, Daten können vom BD-Bus in den Prozessor gelesen werden.
Beim Schreiben ist diese Leitung Low, die Datenrichtung geht vom Prozessor auf den Bus BD0 bis BD7.
Die -ENABLE-Leitung des Datenbuffers wird vom IC UD9, einem 74LS133 angesteuert. Dies IC ist ein NAND-Gatter mit 13 Eingängen. Alle 13 Eingänge müssen High sein, um am Ausgang ein Low zu erzeugen, und nur bei einem Low ist der Buffer überhaupt eingeschaltet.
Das wichtigste Eingangssignal am UD9 ist das Signal S02, der Systemtakt. Sobald dies Signal Low ist, wird der Buffer in den Tri-State-Zustand versetzt. Damit ist der BD-Bus frei für die Arbeit des VIC.

Die Systemsteueranschlüsse

Die R/-W-Leitung des Prozessors wurde gerade beim Datenbusbuffer schon erwähnt. Dieser Anschluß signalisiert mit einem High-Signal, daß der Prozessor einen Lesezugriff auf die Peripheriebausteine ausführen will, bei einem Low wird ein Schreibzugriff ausgeführt.
Damit verbleiben noch drei Prozessorsignale, die wir noch nicht behandelt haben.Das sind die Interrupt-Eingänge -NMI und -IRQ sowie der Eingang -RES.
Zuerst die Interrupt-Eingänge.
Interrupt bedeutet übersetzt Unterbrechung. Mit diesen Eingängen kann der 6502 veranlaßt werden, sein gerade laufendes Programm zu verlassen und auf das Unterbrechungssignal zu reagieren. Wie könnte so eine Unterbrechung in der Praxis aussehen ?
Nun, im einfachsten Fall können wir den Interrupt jede Sekunde einmal auslösen. Der Computer verläßt dann sein momentan laufendes Programm und könnte nun in eine Routine verzweigen, in der eine Uhr programmiert ist. Nach dem Abarbeiten dieser Interruptroutine nimmt der Computer dann sein unterbrochenes Programm wieder auf.
Der 6502 hat zwei Möglichkeiten der Interrupt-Erzeugung. Zum einen ist das der Pin -NMI Non Maskable Interrupt. Wenn dieser Pin Low wird, beendet der Prozessor seinen Maschinensprachebefehl, rettet die momentane Adresse und holt sich von den Adressen $FFFA und $FFFB die Adresse der Interruptroutine. Dieser Vorgang ist durch Programmierung nicht zu verhindern, er findet immer statt.
Erzeugt wird der -NMI durch den -IRQ-Ausgang des 6522 UAB3. Sobald bestimmte programmierbare Ereignisse an diesem IC auftreten, wird der Pin 21 des 6522 Low. Damit wird der -NMI ausgelöst.
Der -NMI-Eingang liegt zusätzlich noch auf dem Kontakt W des Expansion Ports. Damit haben auch externe Baustufen die Möglichkeit, einen Interrupt zu veranlassen.
Auch der -IRQ kann von einer externen Schaltung erzeugt werden. Der Eingang liegt auf Pin 19 des Expansion Ports. Zusätzlich ist noch der -IRQ-Ausgang vom 6522 UA1 mit dem Prozessor verbunden. Auch hier kann wieder das Eintreten

bestimmter Ereignisse zur Erzeugung des Interrupts genutzt werden.
Der Interruptvektor liegt beim -IRQ auf den Adressen $FFFE und $FFFF.
Der -IRQ hat eine Eigenart, über die der -NMI nicht verfügt: er ist softwaremäßig verhinderbar. Damit der Prozessor einen auftretenden -IRQ wahrnehmen kann, muß im Prozessor das Interrupt Disable Flag gelöscht sein.
Der Eingang mit der Bezeichnung -RES wird benutzt, um nach dem Einschalten einen definierten Startpunkt zu haben.
Die Erzeugung dieses Signals wird von dem IC UB6, einem Timer-IC vom Typ 555 und dem nachgeschalteten Inverter UB4 vorgenommen. Im Einschaltmoment laden sich die Kondensatoren C41 und C42 über die Widerstände R24 und R25 auf. Nach wenigen Millisekunden ist die Spannung über dem Kondensator C41 auf ca. 1.7 Volt angestiegen. Damit wird der eigentliche Impuls gestartet. Schlagartig wird der Kondensator C42 über den Pin 7 des UB6 entladen und der Ausgang Pin 3 wird High. Am Ausgang des Inverters liegt damit ein Low-Signal. Nach etwa 1.5 Sekunden ist die Spannung am Kondensator C42 auf ca 3.3 Volt angestiegen. Damit schaltet der Ausgang des Timers auf Low, das Signal -RES wird High. In diesem Umschaltmoment beginnt der Prozessor seine Arbeit.

Der Video Interface Chip 6581

Damit wir Ergebnisse vom Computer zurückbekommen können, muß der Computer über eine Ausgabeeinheit verfügen. Der VC-20 benutzt als Ausgabegerät normalerweise einen Farbfernseher. Alle Signale, die zum Betrieb eines Farbfernsehers nötig sind, werden im sogenannten VIC, dem Video Interface Chip 6561 erzeugt. Die einzige Ausnahme ist das Hochfrequenzsignal, auf dem die Bild- und Farbinformationen aufmoduliert sind. Diese Hochfrequenz wird in dem externen Modulator erzeugt. Alle anderen Signale erzeugt der VIC.
Zusätzlich verfügt dies 40-polige IC über zwei Analog/Digital-Wandler, vier unabhängige programmierbare Tongeneratoren und einen Eingang für einen Light Pen.
Aber zunächst wollen wir uns wieder die Anschlußbelegung des ICs ansehen.

Pin	Bez.	Funktion
1	N.C.	Nicht verwendet
2	COMP COLOR	Ausgang, Farbinformation für Fernsehgerät
3	SYNC & LUM	Ausgang, Syncron- und Helligkeitsinformation für Fernsehgerät
4	R/-W	Eingang, Read/-Write
5	VD11	Datenbit 11
bis		
8	VD8	Datenbit 8
9	BD7	Datenbit 7
bis		

16	BD0	Datenbit 0
17	POT X	Eingang, A/D-Wandler 1
18	POT Y	Eingang, A/D-Wandler 2
19	COMP SND	Ausgang, Sound
20	Vss	Betriebsspannung Masse
21	A0	Adressbit 0
bis		
34	A13	Adressbit 13
35	P01	Taktausgang
36	P02	Taktausgang, nicht verwendet
37	LITE PEN	Eingang für Lichtgriffel
38	02 in	Takteingang Quarzfrequenz
39	01 in	Takteingang Quarzfrequenz
40	Vdd	Betriebsspannung +5V

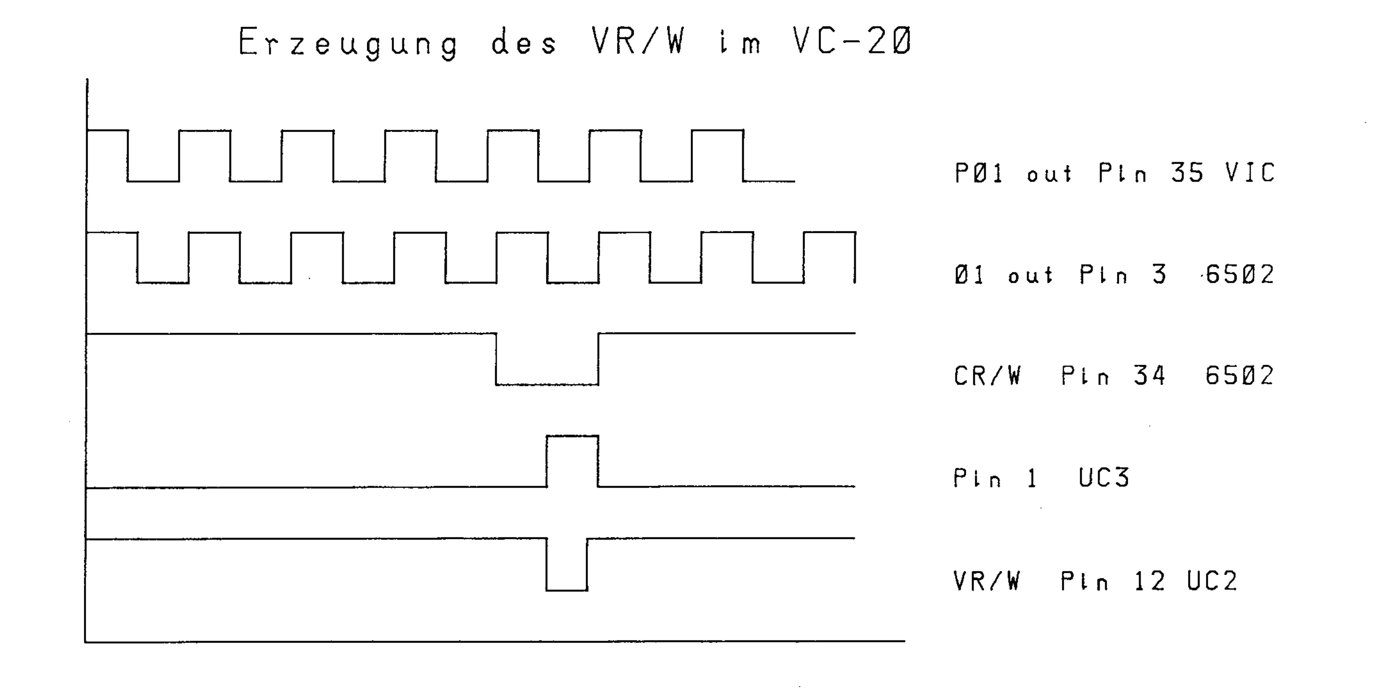
Bei der Betrachtung der Anschlüsse fällt sofort der zusätzliche Datenbus VD8 bis VD11 auf. Damit verfügt der VIC über einen Datenbus von 12 Bit. Das ist auf den ersten Blick sicher etwas ungewöhnlich. Die Antwort ist aber genau so einfach wie überzeugend. Die vier zusätzlichen Bits bestimmen die Farbe der darzustellenden Zeichen. Damit ist auch klar, warum 16 Farben möglich sind. Mit 4 Bits lassen sich genau 16 verschiedene Informationen darstellen.
Diese Datenleitungen sind mit dem RAM UE1 verbunden, dem Farb-RAM. Das Farbram belegt die Adressen $9400 (dez. 37888) bis $97FF (dez. 38911).
Die 8 weiteren Datenleitungen des VIC sind mit dem Datenbus BD verbunden. über den Datenbus-Buffer ist der VIC mit dem Prozessor verbunden. Direkt verbunden ist der VIC mit den Datenleitungen der RAM-Bausteine und dem Charaktergenerator. Dieser Bus hat eine Doppelfunktion. In den Zeiten 02 gleich High kann der Prozessor Daten in das Ram schreiben oder entsprechend auslesen. In der verbleibenden Zeit wird der Bus vom VIC benutzt. Jetzt liest der VIC das Video- und Farb-RAM, und holt sich die zur Darstellung auf dem Bildschirm benötigten Daten aus dem Charakter-ROM.

Auch die Adressleitungen des VIC haben wieder eine Doppelfunktion. In den Zeiten, in denen der Prozessor über die Adressbuffer diesen Bus benutzen kann, werden über diese Leitungen die 16 internen Register des VIC adressiert. Dabei ist eine Besonderheit des VIC zu beachten.
Die Adressleitungen sind nach einem bestimmten Schema aufgeteilt. Die 6 höherwertigen Adressbits A8 bis A13 übernehmen die Funktion der allgemeinen Selektierung des IC. Die Bedingung lautet dabei, daß das Adressbit A13 und die Bits A11 bis A8 Low, daß Adressbit A12 High sein muß. Wenn diese Bedingung stimmt, kann mit den unteren 4 Adressbits A0 bis A3 das gewünschte Register im VIC ausgewählt werden.
Wäre der VIC direkt mit dem Adressbus des Prozezzors verbunden, dann hätte der VIC vier Basisadressen, da die zwei höchsten Adressbits A14 und A15 nicht verwendet werden könnten. Das ist natürlich unerwünscht, zumal die erste Adresse des VIC auf $1000 und die Zweite auf $5000 liegen würde, dem für RAM vorgesehenen Adressbereich.

Darum hat man einen anderen Weg beschritten, die Erzeugung des Adressbits mit einem Dekoder.
Der verwendete Dekoder ist das IC UC5, ein 74LS138. Dieser Dekoder ist an den drei Eingängen mit den Adressbits CA13 bis CA15 verbunden. Entsprechend der Eingangsinformation ist einer der acht Ausgänge Low. Damit ist der gesamte 64K-Adressbereich des Prozessors in acht Blöcke zu jeweils 8K aufgeteilt.
Das Block-Auswahlsignal -BLK4 vom Pin 4 des Dekoders ist im Bereich $8000 bis $9FFF Low. Dieses BLK-Signal wird als Adressbit VA13 benutzt. Damit liegt die Basisadresse des VIC auf $9000 (dez. 36864) fest.
Die zuvor beschriebenen Zustände beziehen sich auf den Zeitpunkt, in dem 02 High ist. In den 02-Low-Zeiten benutzt der VIC den Adressbus VA, um das Video- und Farb-RAM und den Charaktergenerator zu adressieren. Der Adressbus des VIC ist also bidirektional, er arbeitet abhängig von 02 als Eingang oder als Ausgang.

Das Signal VR/-W ist das vom 6502 erzeugte R/-W-Signal. Damit wird am VIC fetgelegt, ob in eins der Register geschrieben, oder daraus gelesen wird. Allerdings kann das R/-W-Signal des Prozessors nicht direkt verwendet werden.
Wie schon bei der Prozessorbeschreibung erläutert, beginnt ein Maschinenzyklus mit der negativen Flanke von 02. Wenn der Prozessor in diesem gerade begonnenen Zyklus einen Schreibzugriff vorbereitet, wird kurz nach Beginn des Zyklus der Anschluß R/-W Low. Zu dieser Zeit belegt aber der VIC den Bus und adressiert beispielsweise das Video-RAM. Wenn jetzt die R/-W-Leitung Low ist, wird dem Video-RAM ein Schreibzugriff vorgetäuscht, der VIC kann aber immer nur lesen. Die Daten im Video-RAM werden dadurch zerstört und der VIC hat keine gültigen Daten erhalten.
Aus diesem Grund wird das R/-W-Signal mit dem Taktsignal 01out des Prozessors und dem Signal P01 des VIC verknüpft. Die genauen Zusammenhänge zwischen den Signalen sehen Sie im Timingdiagramm auf der nächsten Seite.

Die Eingänge POTX und POTY sind die Eingänge der beiden im VIC integrierten Analog/Digitalwandler. Sie sind direkt an die Pins 5 und 9 des Controllerports geführt. Benutzt werden sie in der Regel zum Abfragen der Paddles.
Diese Paddles enthalten zwei veränderbare Widerstände, Potentiometer oder kurz Poti genannt, vergleichbar den Lautstärkereglern an einem Radio. Diese Paddle-Widerstände sind mit dem einen Anschluß an +5V gelegt, der andere Anschluß liegt am entsprechenden Eingang des VIC.
Diese Potis haben an dem einen Anschlag einen kleinen Widerstandswert (ca. 200 Ohm), am anderen Anschlag ist der Widerstandswert relativ hoch (ca 500 KOhm)., Dazwischen ist jeder Wert grundsätzlich möglich. Wie kann der A/D-Wandler nun aus einem Widerstandswert einen Digitalwert erzeugen?
Dazu wird ein kleiner Umweg beschritten. Eine wichtige Rolle spielen dabei die beiden Kondensatoren C20 und C21.


Erzeugung des VR/W im VC-2Ø
PØ1 out Pin 35 VIC
Ø1 out Pin 3 65Ø2
CR/W Pin 34 65Ø2
Pin 1 UC3
VR/W Pin 12 UC2

Wenn der VIC den momentanen Widerstandswert messen will, wird der an Eingang befindliche Kondensator über den Eingang schlagartig entladen. Danach kann der Kondensator sich über den eingestellten Paddle-Widerstand aufladen. Gleichzeitig mit dem Beginn der Ladezeit startet ein Zähler bei 0. Der AD-Eingang überprüft die Spannung am Kondensator und sobald die Spannung höher als 3.5 Volt wird, stoppt der Zähler. Dieser Zählerstand kann aus dem ensprechenden Register des VIC ausgelesen werden.
Wenn der eingestellte Widerstandswert klein ist, wird der Kondensator schnell aufgeladen. Damit stoppt der Zähler schon nach kurzer Zeit und der Wert des A/D-Registers ist klein. Bei einem großen Widerstandswert wird der Kondensator nur langsam aufgeladen, der Zählerstand ist entsprechend größer. Damit ist der Registerinhalt ein direktes Maß des Widerstands oder auch des Drehwinkels.
Diese soeben beschriebenen Meßvorgänge werden alle 0.5 Millisekunden wiederholt. In einer Sekunde wird also 2000 mal der aktuelle Wert in den Registern angezeigt.

An dem Pin 6 des Control Port läßt sich ein sogenannter Light Pen oder Lichtgriffel anschließen. Dieser Pin ist direkt mit dem Anschluß 37 des VIC verbunden.
Das Bild auf dem Fernseher ist aus Punkten zusammengesetzt. Diese Punkte werden von der Bildröhre nacheinander erzeugt. Der VIC 'weiß' zu jeder Zeit, welcher Bildpunkt gerade erzeugt wird.
Dieses Entstehen eines Bildpunktes kann unser Auge nicht wahrnehmen. Uns erscheint das Bild immer gleichmäßig hell.
Mit einem Phototransistor und einer nachfolgenden Verstärkerstufe können die Helligkeitsunterschiede aber nachgewiesen werden. Im Light Pen wird dies Signal so weit verstärkt, das es ein digitales Signal darstellt.
Wenn der Light Pen auf den Bildschirm gehalten wird, entsteht mit dem Erscheinen des Bildpunktes eine negative Flanke am Pin 37 des VIC. Mit diesem Signal wird die augenblickliche Punkteposition in den Light Pen-Registern gespeichert.

An den Anschlüssen 2 und 3 des VIC werden alle für das Erzeugen eines Farbbildes benötigten Signale erzeugt. Dabei sind am Pin 2 die Signale für die Farbinformation, an Pin 3 die Signale für Helligkeit und die Synchronisation vorhanden. Die beiden Signale werden gemeinsam auf die Basis des Transistors Q2 gegeben. Am Emitter des Transistors steht ein Gemisch der beiden Signale zur Verfügung. Diese Signalgemisch wird auf die Kontakte 4 und 5 der 5-poligen DIN-Buchse P4, dem Audio/Video-Port gegeben. An diesen Kontakten kann nun entweder ein Datenmonitor direkt oder über den Modulator der Farbfernseher angeschlossen werden.

Der Pin 19 des VIC ist der Ausgang der vier Tongeneratoren. Da die Signale digital erzeugt werden, ist das Ausgangssignal sehr obertonreich. Das Tiefpassfilter mit den Kondensatoren C9 und C10 und dem Widerstand R8 begrenzt den Frequenzbereich auf die eigentlichen Tonfrequenzen. An der Basis des Transistors Q1 wird dies gefilterte Signal eingespeist,

zusätzlich kann ein externes Audiosignal zugemischt werden. Wenn Sie von dieser Einspeisung eines externen Signals (beispielsweise ein Mikrofonsignal) Gebrauch machen wollen, können Sie das Signal an dem Kontakt Y des Expansion-Ports zuführen. Das Signal sollte eine Größe von ca. 3 Vss haben.
Das Audiosignal wird vom Emitter des Q1 über den Kondensator C11 auf den Kontakt 3 des Video/Audio-Ports gelegt. An diesem Kontakt können Sie direkt einen kleinen Lautsprecher anschließen. Besser wird das Ergebnis natürlich mit einem kleinem Verstärker, da das Ausgangssignal eigentlich nicht für den direkten Betrieb eines Lautsprechers ausgelegt ist.

Damit haben wir die Anschlüsse des VIC und ihre Verwendung kennengelernt. Wie aber werden die Zeichen genau auf dem Bildschirm erzeugt?
Dazu wollen wir uns einmal ein einzelnes Zeichen etwas genauer anschauen. Wie bereits erwähnt, setzen sich die einzelnen Zeichen aus Punkten zusammen. Dieses Punktemuster ist auf einem Farbfernseher nicht mehr zu erkennen. Wenn der VC-20 aber an einen hoch auflösenden Monitor angeschlossen wird, ist das Muster gut zu erkennen. Der Buchstabe A ist folgendermaßen aufgebaut:

```
  1 2 3 4 5 6 7 8
1 . . . * * . . .
2 . . * . . * . .
3 . * . . . . * .
4 . * * * * * * .
5 . * . . . . * .
6 . * . . . . * .
7 . * . . . . * .
8 . . . . . . . .
```

Jedes Zeichen des VC-20 ist in dieser 8 mal 8 Matrix aufgebaut.
Da alleine zur Darstellung der ersten Punktereihe 8 Bits erforderlich sind, benötigt ein ganzes Zeichen somit 8 Bytes. Im Video-RAM wird aber nur ein einziges Byte pro Zeichen abgespeichert.
Dieses Byte kann also nicht direkt das Zeichenpunktemuster enthalten. Das ist aber auch garnicht erforderlich, wir haben ja noch das Charakter-ROM.
In diesem ROM sind die Bitmuster der Zeichen abgespeichert. Jedes Zeichen des VC-20 belegt im Character-ROM 8 Adressen. Die Daten aus dem Video-RAM werden vom VIC nun als Adressbits 0 bis 7 für den Charakter-ROM benutzt. Damit ist die erste Punktereihe adressiert.
Die Adressierung der 7 weiteren Punktebytes übernimmt der VIC jetzt selbstständig mit den Adressleitungen A8 bis A10. Mit jedem Anfang einer neuen Punktereihe auf dem Bildschirm wird die Information auf diesen Leitungen um eins erhöht.
Wenn wir nun einmal ein wenig rechnen, dann haben wir zur Zeichendarstellung 8 Bits zur Verfügung. Damit lassen sich 256 verschiedene Zeichen darstellen. Der VC-20 hat aber inklusive der Reverse-Zeichen einen Vorrat von 512 Zeichen,

genau doppelt so viele.
Der Trick ist nun, daß nicht alle Zeichen zur Zeit darstellbar sind, sondern zwei getrennte Zeichensätze vorhanden sind. Der erste Zeichensatz des VC-20 ist der normale, beim Einschalten vorhandene Satz mit Großbuchstaben und Graphikzeichen. Der zweite, durch Drücken von SHIFT und der Commodore-Taste erreichbare Zeichensatz enthält die Klein- und Großbuchstaben.
Die Umschaltung zwischen diesen Zeichensätzen wird durch das Adressbit A11 des VIC vorgenommen. Ist das Adressbit Low, wird der beim Einschalten vorhandene Groß-Graphiksatz angewählt.

Das RAM, der Arbeitsspeicher.

Ohne Speichererweiterung meldet sich der VC-20 nach dem Einschalten mit der Auskunft, daß 3583 Bytes frei sind, also für Basicprogramm und verwendete Variablen zur Verfügung stehen.
Insgesamt sind im VC-20 5K RAM eingebaut. Die Speicherbausteine sind die ICs U14 und U15 sowie UE2 und UD2. Diese ICs sind sogenannte statische Speicher-ICs.
Die ICs U14 und U15 enthalten jeweils 2048 mal 8 Speicherzellen. Jeweils 8 Speicherplätze belegen zusammen eine Adresse und bilden ein Byte. Die Adressierung der Speicherplätze geschieht über die Adressleitungen A0 bis A11.

Im Gegensatz dazu enthalten die ICs UD2 und UE2 1024 mal 4 Speicherzellen. Um ein Byte zu bekommen, sind zwei ICs erforderlich.
Wenn man die Datenleitungen von UD2 und UE2 verfolgt, sieht man, daß UE2 mit den Datenleitungen BD4 bis BD7 und UD2 mit den Leitungen BD0 bis BD3 verbunden ist. Damit stehen wieder 8 Bits zur Verfügung.
Die Daten- und Adressleitungen der Speicher-ICs sind alle untereinander verbunden. Um jetzt zu unterscheiden, welches IC, also welcher Speicherbereich adressiert wird, ist der Dekoder UC4 nötig.
Dieser Dekoder arbeitet genau so wie der Dekoder UC5, den wir schon in Verbindung mit dem VIC besprochen haben. Beim UC4 ist die Eingangsbeschaltung aber etwas anders.
Damit die Eingänge A,B und C an den Pins 1 bis 3 des Dekoders die Ausgänge beeinflussen können, muß der Eingang G1 High und die Eingänge G2A und G2B Low sein. Diese G-Eingänge stellen eine Freigabe-Funktion dar.
Der Eingang G1 ist mit dem Adressbit VA13 verbunden. Damit ist dieser Eingang immer High, wenn keine Adresse im Bereich $8000 bis $9FFF anliegt. Die Erzeugung von VA13 wurde bereits im Kapitel über den VIC erläutert.
Der Eingang G2B liegt direkt auf Masse, ist also immer Low.
Das Signal an G2A wird vom NOR-Gatter UC3 erzeugt. Der eine Eingang des NOR ist mit dem Signal S01 verbunden. Da dieses Signal aus dem invertierten Signal S02 gewonnen wird, ist es immer dann High, wenn der VIC den Bus belegt. Damit ist

sichergestellt, daß auch der VIC das RAM adressieren kann.
Der zweite Eingang des NOR-Gatters ist über einen Inverter des ICs UC2 mit dem Signal -BLK0 verbunden. Dieses Signal wird vom Dekoder UC5 erzeugt und ist in den Zeiten Low, wenn S02 =High und der Prozessor eine Adresse im unteren 8 K-Bereich auf den Adressbus legt.
Damit ist schon klar, daß der Speicher nur in den unteren 8 K des Prozessor-Adressraums angeordnet sein kann. Aber schauen wir uns jetzt die Eingänge A,B und C des Decoders an. Diese Eingänge sind mit den Adressleitungen VA10 bis VA12 verbunden. Der Decoder arbeitet nun in der Art, daß wenn die drei Adressleitungen Low sind, der Ausgang 0 am Pin 15 Low ist. Wenn VA10 High, die anderen beiden Adressbits aber Low sind, dann ist Ausgang 1 am Pin 14 Low. Alle anderen Ausgänge sind dann High. Auf diese Weise wird der 8 K-Block, der durch das Signal -BLK0 ausgewählt wurde, in kleinere 1 K-Bereiche unterteilt.
Der Pin 15 des Dekoders ist mit dem Pin 8 der beiden Speicher UD2 und UE2 verbunden. Dieser Anschluß an den Speicherbausteinen mit der Bezeichnung -CS, Chip Selekt, bewirkt die grundsätzliche Adressierung des ICs. Sobald dieser Anschluß Low wird, verlassen die Datenleitungen den Tri-State-Zustand und je nach Datenrichtung werden sie zu Eingängen oder Ausgängen. Die Adressierung der einzelnen Bytes geschieht über die Adressleitungen VA0 bis VA9.
Damit beginnt der RAM-Bereich bei $0000 und endet erst einmal bei $03FF. Die Dekoderausgänge 1 bis 3 werden im VC-20 nicht direkt verwendet. Diese Signale werden als -RAM1 bis -RAM3 auf dem Expansion Port auf die Kontakte 14, 15 und 16 geführt. Über diese Signale werden die 3 K RAM in der 3 K-Erweiterung oder im Super-Erweiterungsmodul selektiert.
Die Ausgänge 4 bis 7 werden als -CS für die RAMs U14 und U15 verwendet. Da diese Speicher jeweils 2 K RAM enthalten, werden die UND-Gatter im IC U13 benötigt. Damit werden je zwei Ausgänge mit 1 K-Bereichen verknüpft. An den Ausgängen der UND-Gatter sind somit 2 K-Bereiche selektiert. Das IC U15 liegt im Speicherbereich $1000 bis $17FF, U14 liegt im Speicherbereich $1800 bis $1FFF. Dezimal ausgedrückt ist das der Bereich von 4096 bis 8191.

Die 3 K-Speichererweiterung wurde gerade ja schon erwähnt. Diese Erweiterung schließt die Lücke zwischen den beiden RAM-Bereichen im VC-20. Die 8 K- und 16 K-Erweiterungen schließen sich im Adressbereich nach oben, d.h. zu den höheren Adressen hin an.

Zum Abschluß des Kapitels wollen wir noch die Vorgänge um das Farb-RAM betrachten.
Dies Farb-Ram ist ebenfalls ein 2114 und somit als 1024 mal 4 Bit-Speicher organisiert. Mit diesen 4 Bits sind 16 Kombinationen darstellbar, genau ausreichend für 16 verschiedene Farben.
Der Adressbus dieses RAM ist genau wie die anderen RAMs mit dem VA-Bus verbunden, da ja sowohl der Prozessor als auch der VIC dieses RAM adressieren müssen.
Die Datenleitungen sind aber nicht direkt mit dem BD-Datenbus

verbunden. Hier ist ein Vierfach-Analogschalter, das IC UD1, dazwischengeschaltet.
Diese Analogschalter sind vergleichbar mit einfachen mechanischen Schaltern. Genau wie diese sind Analogschalter bidirektional, d.h. die Daten können ohne Umschaltung der Datenrichtung vom Prozessor sowohl aus dem Farb-RAM gelesen als auch in das RAM geschrieben werden. Ein- und ausgeschaltet werden die Schalter mit einer Spannung an den Steuereingängen Pin 5, 6, 12 und 13. Liegen diese Eingänge auf High, dann sind die Schalter geschlossen, das Farb-RAM liegt mit seinen 4 Datenleitungen auf den unteren vier Datenbits BD0 bis BD3. Ist der Pegel Low, dann sind die Schalter geöffnet und die Datenleitungen liegen nur an den Dateneingängen VD8 bis VD11. Als Steuersignal wird der Systemtakt S02 verwendet.
Der Sinn der Abschaltung wird klar, wenn man sich überlegt, daß wenn der VIC ein Zeichen aus dem Video-RAM liest, dieses Zeichen über den BD-Datenbus in den VIC gelesen wird. Gleichzeitig wird aber die Farbinformation auf dem VD-Datenbus benötigt. Ohne die Abschaltung würde Video-Ram und Farb-Ram gleichzeitig die Datenbits 0 bis 3 belegen, die Datenleitungen würden gegeneinander arbeiten und die Daten wären unbrauchbar.
Etwas aufwendig ist die Adressdekodierung und Erzeugung des -CS-Signals für das Farb-RAM.
Diese Dekodierung wird mit dem dritten Dekoder-IC UC6 im VC-20 vorgenommen. Die Eingänge dieses 74LS138 sind mit den sechs höchsten Adressbits des Prozessors verbunden. Der Eingang G1 ist mit dem Adressbit CA15 verbunden. G1 ist, wie wir schon bei den beiden anderen Dekodern gesehen haben, ein High-aktives Signal, die beiden G2-Eingänge sind Low-aktiv. Die Eingänge können also nur im Adressbereich von $8000 bis $9FFF (dez. 32768 bis 40959) die Ausgänge beeinflussen.
An den eigentlichen Signaleingängen des Dekoders liegen die Adressbits CA10 bis CA12. Somit ändern sich die Ausgänge des Dekoders in 1 K-Schritten. Die im Bereich von $8000 bis $8FFF befindlichen Ausgänge an den Pins 12 bis 15 werden im VC-20 nicht benutzt. Die vier verbleibenden Ausgänge dekodieren den sogenannten I/O-Bereich, das ist der Adressbereich von $9000 bis $9FFF (dez. 36864 bis 40959), in 1 K-Blöcken.
Der Pin 10 des Dekoders ist im zweiten 1 K-Block dieses Adressbereichs Low. Dieses Signal wird zur Erzeugung des -CS-Signals für das Farb-RAM verwendet. Damit liegt das Farb-RAM im Bereich von $9400 bis $97FF.
Das Signal des Dekoders wird zuerst auf den Inverter UC2 gegeben. Am Ausgang des Inverters liegt das Signal COLOR an. Dieses Signal wird auf den Eingang Pin 12 des Busbuffers UE8 geführt. Am Ausgang des Buffers lautet dieses Signal jetzt CEN B, was wohl in etwa COLOR ENABLE BUFFERED bedeuten soll. Dieses CEN B wird nun noch einmal über den Inverter UC2 geführt und dann an den -CS-Eingang des RAMs gegeben.
Jetzt stellt sich die Frage, warum das Signal insgesamt zwei mal invertiert wird. Nach einer zweifachen Invertierung hat jedes Ausgangssignal wieder die selbe Polarität wie das Eingangssignal.
Zur Erklärung müssen wir uns an die zeitliche Doppelbenutzung

von Daten- und Adressbus erinnern. Maßgeblich zum Verständnis ist der Adressbuffer UE8. Wenn der Prozessor den Adressbus belegen kann (02 = High), dann ist der Buffer eingeschaltet. Damit kann der Inverterausgang auf das Signal CEN B einwirken, der Ausgang des Buffers folgt dem Eingang.
Ist dagegen der VIC am Zug und belegt den Bus (02 = Low), dann ist der Buffer im Tri-State-Zustand. Jetzt wird der Eingang Pin 3 des Inverters UC2 über den Widerstand R15 an +5V gelegt und ist damit High. Dadurch ist der Ausgang des Inverters Low, das Farbram ist immer selektiert, wenn der VIC aktiv ist.
Auf diese elegante Art und Weise hat der VIC bei jedem Zugriff auf das Video-RAM die aktuelle, zu dem Zeichen gehörende Farbinformation zur Verfügung.

Das ROM, Das Langzeitgedächtnis des VC-20.

In den ROM-Bausteinen ist alles das gespeichert, was der VC-20 beim Einschalten sofort 'wissen' muß. In diesen ICs ist also der Basic-Interpreter, das Betriebssystem und der Zeichensatz enthalten. Die ROMs werden schon bei der Herstellung programmiert. Die enthaltene Information ist nachträglich nicht mehr zu ändern.
Im VC-20 sind drei ROMs eingebaut. Dies sind die ICs UE11, UE12 und UD7.
UE11 und UE12 enthalten das Basic und das Betriebssystem. UD7 ist das Character-ROM.
Ähnlich wie die RAMs haben auch die ROMs einen -CS-Eingang. Da Basic und Betriebssystem (auch als Kernal bezeichnet) je 8 K groß sind, haben wir die Auswahlsignale schon zur Verfügung. Diese Signale werden vom Dekoder UC5 erzeugt. Dessen Ausgänge selektieren, wie bereits beschrieben, jeweils 8 K-Bereiche. Die -CS-Eingänge sind Low-Aktiv, das bedeutet, daß der Anschluß -CS auf einem Low-Pegel liegen muß, um das IC zu selektieren. Darum können die Dekoderausgänge direkt verwendet werden.
Das Signal -BLK7, Low im Adressbereich $E000 bis $FFFF (dez. 57344 bis 65536), liegt am -CS-Pin 20 vom UE12. Dieses ROM enthält das Betriebssystem.
Das Basic-ROM bekommt sein -CS-Dignal vom Dekoderausgang -BLK6. Dies Signal ist Low, wenn der Prozessor eine Adresse im Bereich $C000 bis $DFFF (dez. 49152 bis 57343) auf den Adressbus legt. Damit ist der Adressbereich des Basic-Interpreters festgelegt.
Der Daten- und Adressbus der beiden ROMs liegt direkt am Prozessor. Durch die Buffer kann der VIC auf diese ICs nicht zugreifen. Das ist aber auch gar nicht nötig.
Anders beim Charakter-ROM. Hier liegen die Adressleitungen auf dem Bus VA, der Datenbus ist der BA-Bus. Damit liegt das ROM sowohl im Zugriffsbereich des Prozessors wie auch des VIC.
Das Charakter-ROM belegt den Adressbereich von $8000 bis $8FFF (dez. 32768 bis 36863). Das ist ein Bereich von 4 K.
Zur Selektion hat das ROM zwei -CS-Eingänge. Diese müssen

beide Low sein, um daß IC zu selektieren. -CS1 ist mit dem Adressbit VA13 verbunden. Wie wir beim VIC gesehen haben, ist dies Adressbit eigentlich der -BLK4-Ausgang des Dekoders UC5. -CS2 ist mit dem Adressbit VA12 verbunden. Wie bereits erwähnt ist VA13 nur im Adressbereich $8000 bis $9FFF Low. Innerhalb dieses Bereichs ist VA12 Low von $8000 bis $8FFF.

Die Schnittstellenbausteine VIA 6522

Die VIAs im VC-20, die ICs UAB1 und UAB3 übernehmen die komplette Ein-Ausgabesteuerung außer der schon beschriebenen Videosteuerung.
Diese VIAs (Versatile Interface Adapter) verfügen über zwei 8-Bit Ports, zwei Intervall-Timer, ein Schieberegister zum einfachen Aufbau einer seriellen Schnittstelle und über Anschlüsse zum Kontrollieren der Datenübertragung auf den 8-Bit Ports, sogenannte Hand-Shake-Leitungen.
Die genaue programm-technische Handhabung dieses ICs können Sie im folgenden Kapitel mit der Registerbeschreibung der Peripheriebausteine nachlesen. Wir wollen uns hier zuerst wieder den verschiedenen Anschlüssen des 6522 zuwenden.

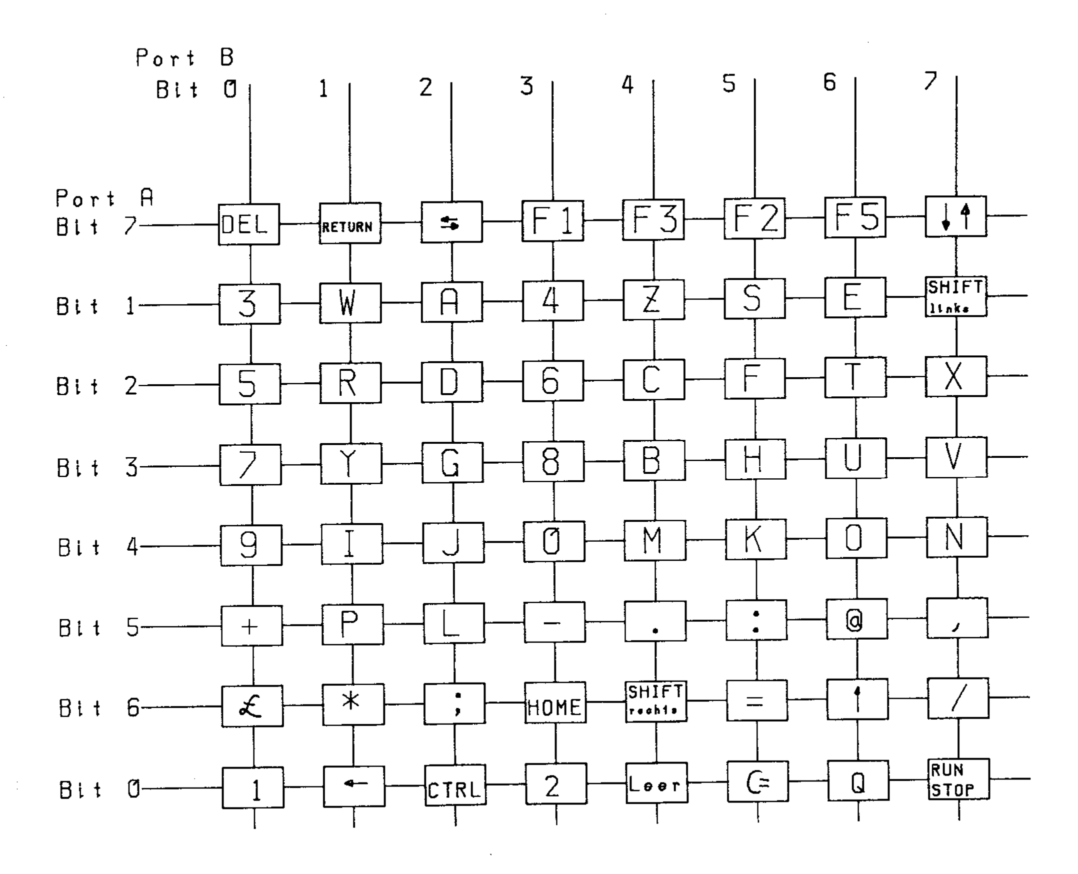
Pin	Bez.	Funktion
1	Vss	Betriebsspannung Masse
2	PA0	Port A bidirektional Bit 0
bis		
9	PA7	Port A bidirektional Bit 7
10	PB0	Port B Bidirektional Bit 0
bis		
17	PB7	Port B bidirektional Bit 7
18	CB1	Port B Controll-Anschluß 1
19	CB2	Port B Controll-Anschluß 2
20	Vcc	Betriebsspannung +5V
21	-IRQ	Ausgang, Interruptsteuerung
22	R/-W	Eingang, Read/-Write Steuerleitung
23	-CS2	Eingang, -Chip Select 2
24	CS1	Eingang, Chip Select 1
25	02	Takteingang, Systemtakt
26	D7	Datenleitung Prozessorbus Bit 7
bis		
33	D0	Datenleitung Prozessorbus Bit 0
34	-RES	Eingang, System-Reset
35	RS3	Eingang, Register-Select 3
bis		
38	RS0	Eingang, Register-Select 0
39	CA2	Port A Controll-Anschluß 2
40	CA1	Port A Controll-Anschluß 1

Der Port A kann je nach Programmierung des 6522 als Ausgabe- oder als Eingabeport arbeiten. Dabei kann die Datenrichtung eines jeden Portbits einzeln programmiert werden. Bei der Verwendung als Eingabeleitung können die Eingabewerte im VIA

gespeichert werden.
Dafür findet der Anschluß CA1 Verwendung. Mit jedem Low-High-Wechsel an CA1 werden die an den als Eingängen verwendeten Portbits anliegende Information im VIA gespeichert. Durch Programmierung sind die beiden CA-Anschüsse aber auch als Interrupteingänge verwendbar.
Port B ist ebenfalls in seiner Datenrichtung an jedem Portbit frei programmierbar. Auch hier können die an den als Eingang programmierten Leitungen liegenden Informationen mit einem Hand-Shake-Signal im VIA gespeichert werden. Zusätzlich ist das Bit PB7 in der Art zu programmieren, daß dieser Anschluß den Ausgang eines der beiden integrierten Timer darstellt. PB6 dagegen ist als Eingang für den anderen Timer programmierbar.
Bei entsprechender Programmierung arbeitet CB1 als Hand-Shake-Eingang. Aber die Anschlüsse CB1 und CB2 können auch in anderen Betriebsarten betrieben werden. Ebenso wie die CA-Leitungen können sie zum einen als Interrupteingänge verwendet werden. Zum zweiten können diese Leitungen aber den Ein- und Ausgang des Schieberegisters darstellen. Damit eignen sich diese Leitungen besonders zur Realisation serieller Datenübertragung.
Der Ausgang -IRQ wird normalerweise zur Erzeugung eines Interrupts verwendet. Dieser Anschluß wird immer dann Low, wenn bestimmte programmierte Ereignisse eintreten. Dies kann zum Beispiel in Verbindung mit den Timern, dem Schieberegister oder auch den Portcontrollanschlüssen geschehen. Der -IRQ-Ausgang des UAB1 ist mit dem -IRQ-Eingang des Prozessors verbunden. Der -IRQ-Ausgang der zweiten VIA UAB3 ist mit dem -NMI-Eingang des Prozessors verbunden.
Der R/-W-Anschluß der VIAs ist direkt mit dem Prozessor verbunden. über diese Leitung signalisiert der Prozessor, ob er Daten in eines der 16 Register des 6522 schreiben oder aus dem VIC lesen will.
Der Datentransfer vom Prozessor zu den VIAs geschieht über den gebufferten Datenbus BD0 bis BD7.
Die Adressierung der 16 internen Register jeder VIA ist etwas aufwendig. Für diesen Zweck verfügt ein 6522 über sechs Pins. Die eigentliche Registerauswahl wird mit den vier RS- oder Register Select-Pins vorgenommen. Diese Anschlüsse sind direkt mit den unteren vier Adressbits des Prozessors verbunden. Damit ergibt sich eine durchgehende Adressierung der Register. Zusätzlich hat jeder 6522 zwei CS- oder Chip Select-Eingänge. Der CS1 ist ein High-aktives Signal, -CS2 ist Low-aktiv. Um also eine VIA zu selektieren, muss das Signal CS1 High, -CS2 dagegen gleichzeitig Low sein.
Die allgemeine Adressdekodierung wird vom Dekoder UC6 vorgenommen. Dieser Dekoder wurde von uns schon bei der Erzeugung des -CS-Signals für das Farb-RAM behandelt. Wie festgestellt, werden durch diesen Dekoder 1 K-Adressblöcke im Bereich $8000 bis $9FFF dekodiert.
Innerhalb dieses Bereichs wird der Ausgang Pin 11 des Dekoders als -CS-Signal für die VIAs verwendet. Dadurch liegen diese ICs im Adressbereich $9000 bis $93FF. Zur Selektion der VIAs ist aber auch noch der CS1-Eingang vorhanden. Um den Hardware-Aufwand gering zu halten, werden

diese Eingänge nicht von einem weiteren Dekoder, sondern von den Adressbits CA4 und CA5 angesteuert. Bei der Dekodierung der Adressbereiche werden die Adressbits CA6 bis CA11 überhaupt nicht benutzt. Dadurch ergibt sich die Möglichkeit, die Basisadresse innerhalb des durch die Dekodierung begrenzten Bereichs frei zu wählen.
Von dieser Möglichkeit muß im VC-20 sogar Gebrauch gemacht werden. Schauen wir uns dafür einmal die vollständige Dekodierung vom VIA UAB3 an.
Das Signal -CS2 ist im Adressbereich $9000 bis $93FF Low. Das erste interne Register kann angesprochen werden, wenn die Adressleitungen CA0 bis CA3 entsprechend den Anschlüssen RS0 bis RS3 Low sind. Die letzte Bedingung ist das Signal CS1. Adressbit CA4 muß somit High sein, um den 6522 zu selektieren. Die erste Adresse, auf der alle Bedingungen erfüllt sind, ist demnach $9010. Da aber die Adressbits CA6 bis CA11 redundant sind, also in der Dekodierung keine Rolle spielen, kann die VIA z.B. auch auf den Adressen $9050, $9110, $9190 und verschiedenen anderen Adressen angesprochen werden.
Der Haken an der ganzen Sache ist nun der VIC. Obwohl er nur 16 interne Register hat, und somit mit dem Adressbereich von $9000 bis $900F auskommen sollte, belegt er doch die 256 Bytes von $9000 bis $90FF. Damit ist also erst ab der Adresse $9100 Platz für die VIAs. Wenn Sie für eigene Experimente mit dem Userport arbeiten wollen, sollten Sie sich daran erinnern, und die Startadressen der VIAs auf $9110 für UAB3 und $9120 für UAB1 festlegen.

Soviel zum Anschuß dieser ICs an den Prozessor. Betrachten wir uns nun einmal, wozu die VIAs überhaupt eingesetzt werden.
Die 8-Bit-Ports A und B von UAB1 werden zum Anschluß der Tastatur verwendet. Die Tastatur ist über den Stecker KEYBOARD CONN mit den beiden Ports verbunden. Die 16 Leitungen der beiden Ports werden in einer 8 mal 8 Matrix zusammengeschaltet. In dieser Matrix sind 64 Positionen möglich. Wenn wir uns die Mühe machen, die Tasten des VC-20 zu zählen, dann kommen wir aber auf 66 Tasten, zwei mehr als mit der Matrix abfragbar sind.
Diese beiden überzähligen Tasten sind die RESTORE- und die SHIFT LOCK-Taste. RESTORE liegt überhaupt nicht in der Matrix, sondern ist separat herausgeführt. Die Taste SHIFT LOCK ist einfach zu der linken SHIFT-Taste parallel geschaltet. Die anderen Tasten haben alle eine bestimmte Position in der Matrix. Diese Positionen sind auf dem Bild auf der nächsten Seite zu sehen.
Die eigentliche Tastaturabfrage läuft folgendermaßen ab.
Der Port A ist als Eingang programmiert. Bei dem 6522 erscheint ein offener Eingang immer als High. Port B ist als Ausgang programmiert. Die Ausgänge sind normalerweise High. Alle 17 Millisekunden werden aber die Ausgänge für ca. 40 Microsekunden Low.


Port B
Bit 0
1
2
3
4
5
6
7
Port A
Bit 7
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 0
DEL
RETURN
F1
F3
F2
F5
3
W
A
4
Z
S
E
SHIFT links
5
R
D
6
C
F
T
X
7
Y
G
8
B
H
U
V
9
I
J
0
M
K
O
N
+
P
L
-
.
:
@
,
£
*
;
HOME
SHIFT rechts
=
↑
/
1
←
CTRL
2
Leer
Q
RUN STOP

Wenn jetzt beispielsweise die Taste 'B' gedrückt wird, dann erscheint dieser kurze negative Impuls an dem Eingang Bit 3 des Port A. Damit ist festgestellt, daß eine Taste gedrückt ist. Um welche Taste es sich handelt, kann aber noch nicht gesagt werden. Darum wird nach dieser allgemeinen Erkennung eine gezielte Suche nach der Tastenposition durchgeführt. Dazu wird jetzt nacheinander jeder Ausgang von Port B kurz Low. Wenn Bit 4 dieses Port Low wird, liegt dieser Impuls über die gedrückte Taste an Port A Bit 3. Damit steht die genaue Position innerhalb der Matrix fest.
Die Bits 3 und 7 des Port B erfüllen im VC-20 eine Doppelfunktion. über Bit 7 wird die Position 'Rechts' des Joysticks abgefragt. Dazu muß natürlich das Bit 7 des Port B auf Eingang programmiert, und die Tastaturabfrage ausgeschaltet werden.
Das Portbit 3 des Port B liegt am Cassetten-Port P2 auf den Kontakten E und 5. über diese Leitung werden die Schreibdaten als serieller Datenstrom zur Datasette geführt.
Auch das Controll-Bit CA1 vom UAB1 liegt am Cassetten-Port P2. Diese Leitung liegt aber an den Kontakten D und 4. über diese Leitung kommen die Lesedaten von der Datasette.
CA2 dieser VIA wird verwendet für den seriellen IEC-Bus. Diese Controll-Leitung liefert über den invertierenden Buffer UB4 das Signal SERIAL CLK OUT am Pin 4 des Serial Port. Dieses Clock-Signal steuert die Geschwindigkeit der Datenübertragung auf dem seriellen Bus.
Auch die Controllbits 1 und 2 des Ports B werden für den seriellen Bus verwendet.
CB1 ist direkt mit dem Pin 1 der 6-poligen Buchse verbunden und ist als Eingang SRQ IN programmiert. SRQ (Service ReQuest) ist ein Signal des 'großen' IEC-Busses. über diese Leitung können angeschlossene Geräte dem Rechner melden, daß Daten vorliegen und 'abgeholt' werden müssen. Allerdings wird dieser Anschluß am VC-20 von keiner uns bekannten Peripherie benutzt.
CB2 liegt ebenfalls über einen invertierenden Buffer im UB4 an der Buchse P3. Dies Bit liefert die seriellen Daten für Floppy, Drucker oder sonstige Peripherie.

Auch die Anschlüsse der zweiten VIA UAB3 werden teilweise doppelt benutzt.
Die Bits 0 und 1 vom Port A werden aber ausschließlich vom seriellen Bus benutzt. Dabei ist das Bit 0 der Eingang SERIAL CLK IN, Bit 1 ist der Eingang SERIAL DATA IN. Um Daten z.B. aus der Floppy in den VC-20 zu bekommen, muß der Bus bidirektional sein. Die verwendeten invertierenden Buffer im IC UB4 lassen aber nur eine Datenrichtung zu. Darum müssen für die Eingänge andere Bits an den VIAs benutzt werden. Das bedeutet aber, daß wenn z.B. die Floppy Daten auf die Leitung SERIAL DATA legt, der Ausgang der Floppy auf dem Ausgang des VC-20 liegt. Da aber die verwendeten Buffer sowohl im VC-20 wie auch in der Floppy sogenannte Open Collector-Ausgänge haben, ist das Zusammenschalten ohne Gefahr für die Ausgänge möglich.
Die verbleibenden 6 Bits von Port A liegen am sogenannten User-Port P1. Dieser User-Port stellt Ihnen die Möglichkeit

zur Verfügung, eigene Steuerungen oder Interfaces ohne großen Hardwareaufwand zu realisieren. Mit diesem Port können Sie beliebig Daten aus dem VC-20 ausgeben oder eingeben. Die Programmierung können Sie der Registerbeschreibung des 6522 in diesem Buch entnehmen. Doch wenden wir uns zuerst wieder den VIAs zu.
über die Bits 2 bis 4 werden die Joystick-Positionen 'Links', 'Oben' und 'Unten' abgefragt. Dabei ist das Bit 2 für die Position 'Oben', Bit 3 für 'Unten' und Bit 4 für die Position 'Links' verantwortlich.
Das Bit 5 trägt die Bezeichnung LITE PEN. Das ist verwunderlich, den der Light Pen-Anschluß gehört ja an den VIC. Der Grund für diese Bezeichnung ist der Controller-Port, an dem sowohl Joystick wie Paddle als auch der Light Pen angeschlossen wird. Um mit der vorhandenen Anzahl von Anschlüssen auszukommen, wurde der Pin 6 des Controller-Port doppelt belegt. Er ist sowohl Eingang für dem Light Pen als auch für den Feuerknopf des Joysticks. Damit ist die Bezeichnung am Bit 5 der VIA UAC3 verständlich. Dieses Bit fragt bei entsprechender Programmierung den Feuerknopf ab.
Die Abfrage, ob an der Datasette eine Taste gedrückt ist, wird mit dem Bit 6 des Port A realisiert. Dabei ist aber zu beachten, daß der VC-20 nicht unterscheiden kann, welche Taste gedrückt ist, da in der Datasette alle Tasten gleichzeitig abgefragt werden.
Auch das Bit 7 des Ports A wird für den seriellen Bus verwendet. Es trägt die Bezeichnung SERIAL ATN IN.
ATN ist die Abkürzung für Attention, auf Deutsch so viel wie ACHTUNG. Mit diesem Signal wird den Peripheriegeräten signalisiert, daß der VC-20 Anweisungen an die angeschlossenen Geräte senden will. Da der VC-20 innerhalb einer Anlage immer die Aufgabe eines Controllers hat, verfügt er auch über keinen entsprechenden Eingang.
Das Signal SERIAL ATN IN liegt nur auf dem User-Port. Dieses ATN IN wird von keiner Erweiterung der Firma Commodore genutzt. Diese Beschaltung hat aber den Vorteil, daß das Portbit auch am User-Port zur Verfügung steht.
Bei der Benutzung dieses Bits müssen Sie nur darauf achten, daß Sie dieses Bit nur als Ausgang verwenden können, und zusätzlich die Information durch den Buffer UB4 invertiert wird.
Das Controll-Bit CA 1 hat eine ganz besondere Bedeutung. An diesem Pin von UAB3 ist die RESTORE-Taste angeschlossen. Wenn diese Taste nicht gedrückt wird, legt der Widerstand R16 den Anschluß auf einen High-Pegel. Wird die Taste aber gedrückt, dann liegt der Anschluß auf Masse und in der VIA wird ein Interrupt erzeugt. Der Anschluß -IRQ der VIA wird sofort Low und erzeugt damit am Prozessor einen -NMI.
CA2, das zweite Controll-Bit des Port A, wird wieder für die Datasette benötigt. über dieses Bit wird der Recorder-Motor gesteuert.
Diese Motorsteuerung wollen wir uns noch einmal etwas genauer ansehen. Wichtig hierfür sind die Transistoren Q3 und Q4. Wenn das Bit CA2 High ist, wird die Basis des Transistors Q3 über den Widerstand R20 angesteuert; der Transistor ist durchgeschaltet. Damit ist aber die Zenerdiode CR1

kurzgeschlossen, die Basis des zweiten Transistors Q4 liegt auf Masse. Damit ist dieser zweite Transistor gesperrt und der Rekordermotor bekommt keinen Strom; der Motor steht. Wenn das Bit CA2 aber Low wird, dann sperrt der Transistor Q3. Jetzt kann sich über der Diode CR1 die sogenannte Zenerspannung aufbauen. Fertigungsbedingt hat die an dieser Stelle eingesetzte Diode eine Zenerspannung von ca. 6.8 Volt. Damit wird der Transistor Q4 angesteuert. Am Emitter dieses Transistors stellt sich jetzt eine Spannung ein, die etwa 0.8 Volt unter der Spannung an der Basis von Q4 liegt (ca. 6 Volt). Diese Spannung ist relativ unabhängig von der Belastung durch den Motor. Dadurch läuft die Datasette immer mit annähernd gleicher Geschwindigkeit, eine für den störungsfreien Betrieb sehr wichtige Voraussetzung.
Der vollständige 8-Bit-Port B der VIA UAB3 und die dazugehörenden Controll-Bits 1 und 2 liegen ausschließlich am User-Port. Bei diesen Portbits brauchen Sie keine Rücksicht auf eine eventuelle Benutzung durch das Betriebssystem zu nehmen. Aber auch hierbei gibt es wieder eine Ausnahme. Wenn Sie nämlich Ihren VC-20 mit einer RS-232 Schnittstelle ausrüsten, dann werden die freien Leitungen des User-Port für den Betrieb dieser Schnittstelle verwendet. Das ist aber nicht sonderlich schlimm, da die Kontakte des User-Port dann ja sowieso von der Schnittstelle belegt werden und Ihnen nicht mehr zur Verfügung stehen.
Insgesamt stehen Ihnen am User-Port also 10 ganz frei programmierbare Leitungen und 6 bedingt freie Leitungen zur Verfügung.
Bedingt Frei bedeutet, daß Sie bei der Benutzung dieser Bits eine mögliche Kollision mit dem Betriebssystem vermeiden müssen.
Zusätzlich liegt am User-Port am Kontakt 3 noch das Signal -RES, um eventuelle externe Schaltungen beim Einschalten in einen Grundzustand zu versetzen und am Kontakt 2 die stabilisierte Betriebsspannung von +5V zur Versorgung kleinerer Schaltungen. Größere Schaltungen können Sie versorgen, wenn Sie die 9 Volt Wechselspannung an den Kontakten 10 und 11 gleichrichten und stabilisieren. Bedenken Sie aber, daß der maximale Wechselstrom von der Benutzung des Recorders abhängt.
Was können Sie aber nun mit dem User-Port anfangen?
Einige denkbare Anwendungen sind die Realisation einer Centronics-Schnittstelle, der Anschluß eines EPROM Programmiergerätes, die Abfrage von Analog/Digital Wandlern oder umgekehrt die Ansteuerung von Digital/Analog Wandlern, die Steuerung von Motoren, die Abfrage von Endschaltern und, und, und....
Hier können Sie Ihrer Phantasie ganz und gar freien Lauf lassen. Vielleicht fällt Ihnen ja noch etwas ganz außergewöhnliches ein. Wie wäre es mit einer Steuerung für die Stereoanlage mit eingebauter Alarmanlage für das Haus?

Registerbeschreibung des VIC 6561

Der VIC verfügt über 16 Register, die im Folgenden beschrieben werden.
Die Basisadresse des VIC im VC-20 ist 36864 ($9000)

REG 0 Bildabstand vom linken Rand
Die Bits 0-6 bestimmen den Abstand des Bildrahmens vom linken Rand. Das Erhöhen dieses Registers um 1 verschiebt das Bild um jeweils vier Punkte nach rechts.
Bit 7 legt fest, ob das Bild im Zeilensprungverfahren (=1) dargestellt wird oder nicht (=0). Das Zeilensprungverfahren besteht einfach darin, daß ein Bild aus zwei Halbbildern zusammengesetzt wird, und zwar aus einem Bild mit allen geraden Rasterzeilen und einem mit allen ungeraden Rasterzeilen. Bei Manipulation dieses Bits kann am Fernsehschirm keine Wirkung beobachtet werden.

REG 1 Bildabstand vom oberen Rand
Mit diesem Register wird der Abstand des Bildrahmens vom oberen Rand bestimmt. Das Erhöhen dieses Registers um 1 verschiebt das Bild um jeweils zwei Punkte nach unten.

Anmerkung zu REG 0&1
Bei Spielmodulen gibt es nach dem Einschalten oft die Möglichkeit, mit Hilfe der Cursorsteuertasten das Bild zu zentrieren.

REG 2 Anzahl der Zeichen/Zeile
Die Bits 0-6 legen fest, wieviel Zeichen pro Zeile dargestellt werden sollen.
Bit 7 ist Bestandteil von REG 5 und wird dort beschrieben.

REG 3 Anzahl der Zeilen
Bit 0 bestimmt die Darstellungsart eines Zeichens, entweder als 8x8-Matrix (=0, Normalfall) oder als 16x8-Matrix (=1).
Mit den Bits 1-6 kann die Anzahl der Zeilen pro Bild festgelegt werden.
Bit 7 ist Bestandteil von REG 4.

Anmerkung zu REG 2&3
Wenn Sie die Anzahl der Zeichen oder Zeilen verändern wollen, so beachten Sie bitte, daß andere als die nach dem Einschalten in diesen Registern diesbezüglich enthaltenen Werte (durch PEEK zu ermitteln) vom Betriebssystem nicht unterstützt werden. Das Betriebssystem geht z.B. bei der Organisation von Basiczeilen davon aus, daß der Bildschirm auf 23 Zeilen mit je 22 Zeichen eingestellt ist. Sie können auch mit den PRINT-Befehlen die Grenzen des Videoram nicht überschreiten.

REG 4 Laufende Rasterzeile

Die Bits 0-7 geben zusammen mit Bit 7 aus REG 3 (dieses bildet dann Bit 0 der Gesamtzahl) die Rasterzeile wieder, die gerade vom Strahl der Bildröhre überstrichen wird. Der Wert kann beispielsweise dazu benutzt werden, an einer bestimmten Position des Bildes die Basisadresse des Videorams (s. REG 5) zu ändern (Split-Screen-Verfahren). Allerdings dürfte hierzu nur ein Programm in Maschinensprache schnell genug sein.

REG 5 Basisadressen

Die Bits 0-3 bestimmen die Basisadresse des Zeichengenerators. Auf die aktuelle Speicheradresse bezogen sind das die Adressbits 10-13. Im VC-20 ist der eingebaute Zeichengenerator solange aktiviert, wie Bit 2&3 =0 sind. Bei Bit 3 =1 wird der Zeichengenerator im Ram gesucht, abhängig von den Bits 0-2, mit deren Hilfe sich der Zeichengenerator ab der Speicheradresse 0 in Schritten von 1K verschieben läßt. So können Sie im Ram Ihren eigenen Zeichensatz aufbauen. Achten Sie hierbei jedoch darauf, daß Sie dem Betriebssystem nicht ins Gehege geraten (s. Speicherbelegung und Zero-Page).
Die Bits 4-7 bilden zusammen mit Bit 7 aus REG 2 die Startadresse des Videorams. Bezogen auf die aktuelle Speicheradresse sind das die Adressbits 13-9. Da Adressbit 13 im VC-20 aus Hardwaregründen immer =1 sein muß, ergibt sich eine Verschiebemöglichkeit des Videorams innerhalb der unteren 4k des Speichers in Schritten von 512 Bytes. Achten Sie auch hierbei auf Konflikte mit dem Betriebssystem.

REG 6 Lightpenposition horizontal

Dieses Register enthält eine zwischengespeicherte Punktposition (s. Anmerkung).

REG 7 Lightpenposition vertikal

Dieses Register enthält eine zwischengespeicherte Rasterzeilenposition.

Anmerkung zu REG 6&7

Die augenblickliche Position des Bildröhrenstrahles wird dann in den REG 6&7 gespeichert, wenn am Anschluß LIGHTPEN (Pin 6 des Controlport oder Pin 7 des Userport) ein übergang von 1 nach 0 auftritt. Diese Möglichkeit ist zum Malen mit einem Lichtgriffel oder für Schießspiele am Bildschirm interessant.

REG 8 gibt den Wert am Analogeingang X (Pin 9 des Controlport) wieder.

REG 9 gibt den Wert am Analogeingang Y (Pin 5 des Controlport) wieder.

REG 10 Tongenerator 1

Die Bits 0-6 bestimmen die Tonhöhe des Generators.
Bit 7 =1 schaltet den Generator ein, =0 aus.

REG 11 Tongenerator 2
Bedeutung der Bits wie REG 10

REG 12 Tongenerator 3
Bedeutung der Bits wie REG 10

REG 13 Rauschgenerator
Bedeutung der Bits wie REG 10

REG 14 Lautstärke/Farbauswahl
Die Bits 0-3 regeln kollektiv die Lautstärke aller Generatoren.
Die Bits 4-7 bestimmen eine Farbe im Zusammenhang mit dem Multicolor-Mode (s. Betriebsarten des VIC).

REG 15 Rahmen-/Hintergrundfarbe
Mit den Bits 0-2 kann eine Rahmenfarbe ausgewählt werden.
Bit 3 =1 läßt verschiedenfarbige Zeichen auf einer gemeinsamen Hintergrundfarbe erscheinen, während Bit 3 =0 die Zeichen gleichfarbig (Farbe aus Bit 4-7) auf verschiedenfarbigem Hintergrund darstellt. Dieses Bit hat im Multicolor-Mode keine Bedeutung.
Die Bits 4-7 bestimmen die Hintergrundfarbe.

Die Betriebsarten des VIC

Der VIC kennt zwei Arten der Darstellung, nämlich Hi-Res-Mode und Multicolor-Mode.
Die beiden Betriebsarten unterscheiden sich lediglich in der Interpretation des Inhaltes des Zeichengenerators.
Ausgewählt werden die Betriebsarten durch den Zustand des höchstwertigen Bits (Bit 3) im Farbram (im Farbram sind ja jeder Position des Videorams vier Bits zugeordnet).

Hi-Res-Mode (Farbbit 3 =0)

Diese Betriebsart ist normalerweise nach dem Einschalten des VC-20 (oder nach STOP/RESTORE) eingestellt.
Hierbei besteht eine eins zu eins Zuordnung der Bits des Zeichengenerators mit den Punkten auf dem Bildschirm. Das bedeutet, daß alle 1-Bits des Zeichengenerators in einer von den Farbbits 0-2 abhängigen Farbe dargestellt werden, und alle 0-Bits in der Hintergrundfarbe (REG 15). Beachten Sie, daß Bit 3 in REG 15 diese Verhältnisse umkehren kann.
Normalerweise besteht das dargestellte Bitmuster aus lesbaren Zeichen, die im Hardware-Charactergenerator des VC-20 festgelegt sind, jedoch ist ohne weiteres ein Zeichnen auf dem Bildschirm möglich, wenn Sie die Startadresse des Zeichengenerators ins Ram verlegen. Die Bildschirmspiele arbeiten auch so.

Multi-Color-Mode (Farbbit 3 =1)

Diese Betriebsart ist ein wenig trickreicher als die vorige. Hier werden zur Darstellung eines Punktes (in Wirklichkeit sind es zwei) gleich zwei Bits aus dem Zeichengenerator herangezogen.
Die Farbe des Punktepaares wird indirekt durch eben diese zwei Bits bestimmt, mit deren Hilfe eine aus vier Farbquellen ausgewählt wird. Da jedoch zur Farbbestimmung eines Punktes zwei Bits aus dem Zeichengenerator herangezogen werden, beträgt die horizontale Auflösung nur die Hälfte des Hi-Res-Mode, d.h. ein 8x8-Block (die Größe eines 'normalen' Zeichens) ruft auf dem Bildschirm nur einen 8x4-Block hervor, wobei die Punkte doppelt breit sind, also dennoch einen vollständigen Zeichenplatz belegen.
Auf diese Weise ist der Speicherbedarf für beide Betriebsarten gleichgroß.
Die beiden Bits des Zeichengenerators bestimmen, woher die Farbe für einen solchen doppelt breiten Punkt genommen werden soll. Folgende Kombinationen sind möglich:

00 REG 15 Bit 4-7 (Hintergrundfarbe)

01 REG 15 Bit 0-2 (Rahmenfarbe)

10 Farbbits 0-2 aus dem korrespondierenden Farbram

11 REG 14 Bit 4-7

Es muß noch einmal betont werden, daß die beiden Bits aus dem Zeichengenerator nicht unmittelbar die Farbe bestimmen, sondern nur die Quelle angeben, aus der die Farbe entnommen werden soll. Wenn Sie die Farbe eines Bildschirmpunktes bei einer gegebenen Bitkombination, z.B. 00, ändern wollen, so müssen Sie die Bits 4-7 in REG 15 manipulieren.

Registerbeschreibung der VIA 6522

Die VIA 6522 verfügt über 16 Register, mit deren Hilfe zwei 8-Bit-Datenports, zwei 16-Bit-Timer, ein Schieberegister und diverse Steuerleitungen gehandhabt werden.
Die Basisadressen der beiden im VC-20 vorhandenen VIAs sind 37136 ($9110, IEC-Bus und Userport) und 37152 ($9120, Tastatur)
Im Folgenden sind die Register ausführlich beschrieben.

REG 0 Datenregister B (ORB)
Dieses Register gibt den Zustand des Datenport B wieder. Es kann sowohl geladen als auch gelesen werden.
Achtung: Ein Zugriff auf dieses Register setzt die Bits 3&4 des IFR (REG 13) zurück und kann, abhängig vom Zustand der Bits 5-7 im PCR (REG 12), die Leitung CA2 beeinflussen!

REG 1 Datenregister A (ORA)
Hiermit wird der Datenport A gehandhabt, Zugriffsart Lesen/Schreiben.
Achtung: Die Bits 0&1 des IFR (REG 13) werden zurückgesetzt. Außerdem kann, abhängig vom Zustand der Bits 1-3 im PCR (REG 12), die Leitung CA2 beeinflußt werden.

REG 2 Datenrichtung Port B (DDRB)
Mittels dieses Registers können die Leitungen des Datenport B individuell auf Eingabe oder Ausgabe programmiert werden. Ist ein Bit dieses Registers =0, so steht die korrespondierende Datenleitung auf Eingabe; ist es =1, steht die Leitung auf Ausgabe.

REG 3 Datenrichtung Port A (DDRA)
Bedeutung wie REG 2, allerdings für den Datenport A.

REG 4 Timer 1 LO (T1CL)
Beim Lesezugriff gibt dieses Register den Zustand des niederwertigen Teiles von Timer 1 wieder. Beim Laden wird zunächst nur ein Zwischenspeicher mit dem neuen Wert versorgt. Die Übernahme in den Zähler erfolgt erst mit Schreiben des höherwertigen Bytes (REG 5).

REG 5 Timer 1 HI (T1CH)
Beim Lesen erhalten Sie den höherwertigen Teil von Timer 1. Ein Schreibzugriff lädt sowohl den Zwischenspeicher als auch den Zähler. Gleichzeitig wird das niederwertige Byte aus dem Zwischenspeicher in den Zähler übernommen.
Achtung: Bit 6 des IFR (REG 13) wird beim Schreiben zurückgesetzt!
Abhängig von Bit 6&7 des ACR (REG 11) kann die Leitung PB7 und das Bit 6 des IFR (REG 13) beeinflußt werden.

ACR6&7 =00:
Beim Nulldurchgang des Zählers wird IFR6 gesetzt. Der Timer zählt zwar nun erneut bis Null, jedoch kann IFR6 erst wieder nach dem Schreiben von REG 5 gesetzt werden.
ACR6&7 =01:
Wie vor, jedoch wird PB7 beim Laden von REG 5 =LO und beim nächsten Nulldurchgang =HI. <u>PB7 steht unabhängig von DDRB auf Ausgang!</u>
ACR6&7 =10:
Der Timer zählt zyklisch vom einmal geladenen Wert auf Null, jedoch wird IFR6 nur beim ersten Nulldurchgang nach Laden von T1CH gesetzt. IFR6 kann durch Lesen von T1CL, Schreiben von T1CH oder explizit, wie bei REG 13 beschrieben, rückgesetzt werden.
ACR6&7 =11:
Wie vor, jedoch kippt PB7 bei jedem Nulldurchgang in die jeweils andere Lage.

REG 6&7 Timer 1 LO&HI (T1LL&T1LH)
Vom Zugriff auf diese Register sind sowohl beim Schreiben als auch beim Lesen grundsätzlich nur die Zwischenspeicher der jeweiligen Zähler betroffen. Die Zähler selbst werden nicht verändert. Interessant ist dieser Effekt, wenn die Timer im FREE-RUNNING-MODE (REG 11 Bit 6 =1) betrieben werden. Hierbei wird bei jedem Nulldurchgang des Zählers dieser automatisch mit dem Inhalt des Zwischenspeichers geladen.
<u>Achtung:</u> Auch beim Schreibzugriff auf REG 6 wird Bit 6 des IFR (REG 13) zurückgesetzt!

REG 8&9 Timer 2 LO&HI (T2CL&T2CH)
Für diese Register trifft bezüglich Lesen und Laden sinngemäß die Beschreibung der REG 4&5 zu. Allerdings gibt es hier nur zwei Betriebsarten, abhängig von Bit 5 im ACR (REG 11)
ACR5 =0:
Sinngemäß wie bei Timer 1, Betriebsart 000 beschrieben; allerdings ist hier Bit 5 des IFR (REG 13) vom Nulldurchgang betroffen.
ACR5 =1:
Ein in den Timer 2 geladener Wert wird durch von außen an PB6 angelegte Impulse auf Null gezählt, wobei beim Nulldurchgang IFR5 gesetzt wird. Der Timer zählt nun erneut auf Null, jedoch kann IFR5 erst wieder beim dem Laden von REG 9 folgenden Nulldurchgang gesetzt werden.
<u>Achtung:</u> Ein Schreibzugriff auf REG 9 setzt Bit 5 des IFR (REG 13) zurück!

REG 10 Schieberegister (SR)
Das Schieberegister kann ohne weiteres geschrieben und gelesen werden. Die hierbei ausgelösten Vorgänge sind abhängig vom Zustand der Bits 2-4 des ACR (REG 11).
ACR2-4 =000:
Das Schieberegister kann ohne weitere Auswirkungen geschrieben und gelesen werden.

ACR2-4 =100:
Die an CB2 anliegenden Daten werden mit einer durch T2CL (REG 8) bestimmten Geschwindigkeit in SR hineingeschoben. Der Schiebetakt kann durch die Formel CLK/2/n berechnet werden, wobei CLK die Systemtaktfrequenz (im VC-20 ca. 1,02MHz) und n der nach REG 8 geladene Wert ist. Dieser Schiebetakt steht auch an CB1 zur evtl. Synchronisation der Datenquelle zur Verfügung. Nach acht Impulsen wird der Schiebevorgang unterbrochen und erst fortgesetzt, wenn SR gelesen wurde. Außerdem wird Bit 2 des IFR (REG 13) gesetzt.

ACR2-4 =010:
Der Schiebetakt wird hier durch den Systemtakt erzeugt und beträgt CLK/2. Sonst wie vor.

ACR2-4 =110:
Der Schiebetakt muß in dieser Betriebsart von außen an CB1 zugeführt werden. Nach acht Takten wird auch hier IFR2 gesetzt, jedoch der Schiebevorgang nicht unterbrochen.

ACR2-4 =001:
Der Inhalt des Schieberegisters wird mit einer durch den Systemtakt und den Inhalt von REG 8 bestimmten Geschwindigkeit nach CB2 hinausgeschoben. Dieser Vorgang läuft zyklisch, d.h. die aus SR hinausgeschobenen Bits werden wieder hineingeleitet. Der Schiebetakt ist wieder an CB1 verfügbar.

ACR2-4 =101:
Der Schiebevorgang wird nach acht Takten abgebrochen und erst nach Laden des SR fortgsetzt. Außerdem wird IFR2 gesetzt. Sonst wie vor.

ACR2-4 =011:
Der Schiebetakt beträgt CLK/2. Sonst wie vor.

ACR2-4 =111:
Die Schiebeimpulse müssen von außen an CB1 angelegt werden. Nach acht Takten wird zwar IFR2 gesetzt, jedoch der Schiebevorgang nicht unterbrochen.

Achtung: Für alle Betriebsarten gilt, daß nach acht Impulsen, gleichgültig ob intern erzeugt oder von außen angelegt, das Bit 2 im IFR (REG 13) gesetzt wird. Gelöscht wird dieses Bit durch Schreiben oder Lesen des SR.

REG 11 Hilfssteuerregister (ACR)

Die Bits dieses Registers werden hier nur soweit beschrieben, wie sie nicht bereits bei den Timern und dem Schieberegister behandelt wurden.

Bit 0 =0: Die aus ORA gelesenen Daten entsprechen dem aktuellen Zustand der Leitungen PA0-7.
=1: Der beim Setzen von IFR1 durch CA1 an PA0-7 vorhandene Zustand wird bis zum Löschen von IFR1 (REG 13) im ORA festgehalten.
Bit 1: Sinngemäß wie oben, jedoch sind hier ORB, PB0-7, CB1 und IFR4 betroffen.
Bit 2-4: Steuerung des Schieberegisters SR (REG 10)
Bit 5: Steuerung von Timer 2 (REG 8&9)
Bit 6-7: Steuerung von Timer 1 (REG 4&5)

REG 12 Port-Steuerregister (PCR)

Bit 0 =0: Eine fallende Flanke an CA1 setzt IFR1
=1: Eine steigende Flanke setzt IFR1
Bit 1-3=000: Eine fallende Flanke an CA2 setzt IFR0. IFR0 wird beim Zugriff auf ORA oder explizit zurückgesetzt.
=100: Wie oben, jedoch kann IFR0 nur explizit rückgesetzt werden.
=010: Wie bei =000, jedoch steigende Flanke.
=110: Wie bei =100, jedoch steigende Flanke.
=001: Zugriff auf ORA setzt CA2=LO. Nach einer aktiven Flanke an CA1 wird CA2 wieder HI.
=101: Nach einem Zugriff auf ORA wird CA2 für die Dauer eines Systemtaktes LO.
=011: CA2 wird LO.
=111: CA2 wird HI.
Bit 4 =0: Eine fallende Flanke an CB2 setzt IFR4.
=1: Eine steigende Flanke setzt IFR4.
Bit 5-7=000: Eine fallende Flanke an CB2 setzt IFR3, welches mittels Zugriff auf ORB oder explizit rückgesetzt werden kann.
=100: Wie oben, jedoch kann IFR3 nur explizit rückgesetzt werden.
=010: Wie bei =000, jedoch steigende Flanke.
=110: Wie bei =100, jedoch steigende Flanke.
=001: Ein Schreibzugriff auf ORB setzt CB2=LO. Eine aktive Flanke an CB1 setzt CB2 wieder HI.
=101: Nach einem Schreibzugriff auf ORB wird CB2 für die Dauer eines Systemtaktes LO.
=011: CB2 wird LO.
=111: CB2 wird HI.

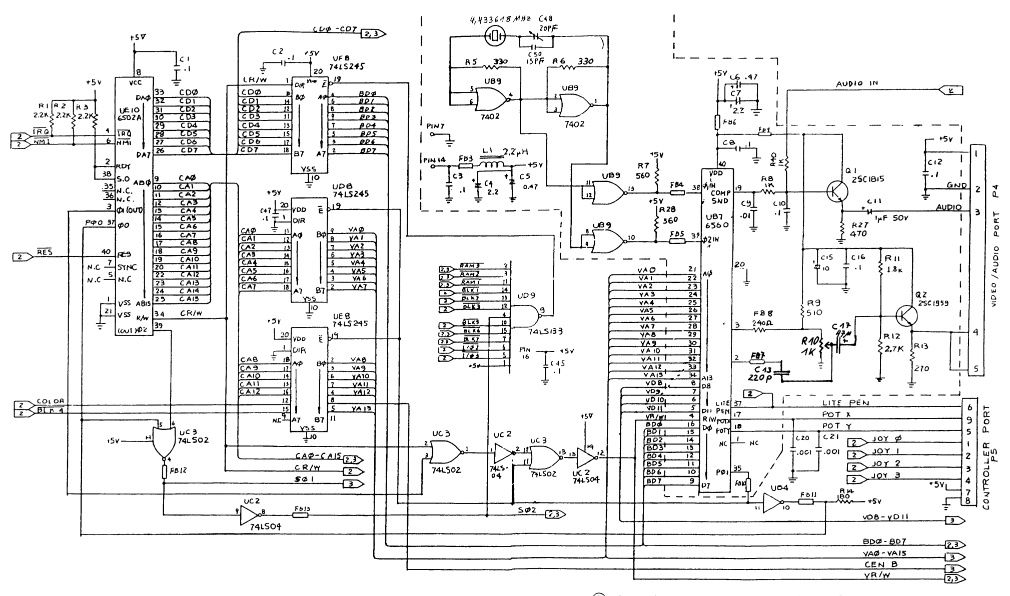
REG 13 Interrupt-Flag-Register (IFR)
Die einzelnen Bits dieses Registers signalisieren das Eintreffen von Ereignissen, deren Eintrittsbedingungen hier nur noch global beschrieben werden, da sie im einzelnen bereits an ihren Ursprüngen erläutert wurden.
Bit 0: wird mit einer aktiven Flanke an CA2 gesetzt und mit einem Zugriff auf ORA (REG 1) rückgesetzt.
Bit 1: wird mit einer aktiven Flanke an CA1 gesetzt und mit einem Zugriff auf ORA (REG 1) rückgesetzt.
Bit 2: wird nach acht Schiebeimpulsen gesetzt und mit einem Zugriff auf SR (REG 10) rückgesetzt.
Bit 3: wird mit einer aktiven Flanke an CB2 gesetzt und mit einem Zugriff auf ORB (REG 0) rückgesetzt.
Bit 4: wird mit einer aktiven Flanke an CB1 gesetzt und mit einem Zugriff auf CB2 (REG 0) rückgesetzt.
Bit 5: wird beim Nulldurchgang von Timer 2 gesetzt und mit einem Lesezugriff auf T2CL (REG 8) oder einem Schreibzugriff auf T2CH (REG 9) rückgesetzt.
Bit 6: wird beim Nulldurchgang von Timer 1 gesetzt und mit einem Lesezugriff auf T1CL (REG 4) oder einem Schreibzugriff auf T1CH (REG 5) rückgesetzt.
Bit 7: wird gesetzt, wenn mindestens ein gesetztes Bit dieses Registers mit dem korrespondierenden gesetzten Bit im IER (REG 14) zusammentrifft. Dieses Bit gibt den Zustand der Leitung IRQ wieder.
Achtung: Die Bits 0-6 dieses Registers können auch explizit dadurch rückgesetzt werden, indem man das

Register mit einem Byte lädt, dessen gesetzte Bits (=1) mit dem zu löschenden Bit dieses Registers korrespondieren.

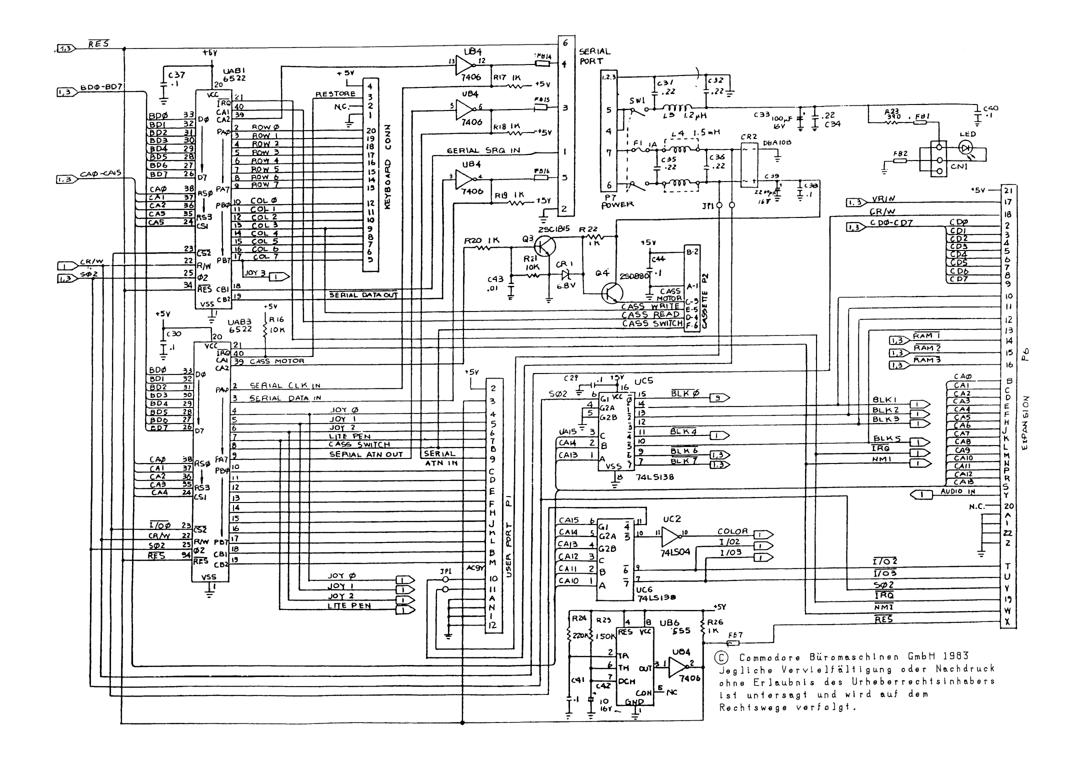
REG 14 Interrupt-Enable-Register (IER)
Die Bits dieses Registers korrespondieren mit den Bits des IFR. Beeinflussung der Leitung IRQ wie unter IFR 7 beschrieben.
Die Bits dieses Register lassen sich setzen, indem man das Register mit einem Byte lädt, dessen gesetzte Bits mit dem zu setzenden Bit dieses Registers übereinstimmen, wobei das höchstwertige Bit (Bit 7) =1 sein muß. Die 0-Bits im Datenbyte lassen die korrespondierenden Bits dieses Registers unverändert. Sollen einzelne Bits rückgesetzt werden, so ist der Vorgang wie oben beschrieben, jedoch muß hier das höchstwertige Bit des Datenbytes (Bit 7) =0 sein.

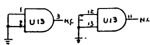
REG 15 Datenregister A (ORA)
Der Inhalt dieses Registers spiegelt die Datenleitungen PA 0-7 wieder, wie unter REG 1 beschrieben, jedoch hat der Zugriff in diesem Falle keinerlei Auswirkung auf die Steuerleitungen oder das IFR.

SCHALTPLAN



UE10
6502A
UF8
74LS245
UD8
74LS245
UE8
74LS245
UD9
74LS133
UB9
7402
UB7
6560
UC3
74LS02
UC2
74LS04
UD4
4,433618 MHz
AUDIO IN
VIDEO/AUDIO PORT P4
CONTROLLER PORT P5
LITE PEN
POT X
POT Y
JOY Ø
JOY 1
JOY 2
JOY 3
CDØ-CD7
CAØ-CA15
CR/W
SØ1
SØ2
VDB-VD11
BDØ-BD7
VAØ-VA15
CEN B
VR/W
COLOR
BLK 4
IRQ
NMI
RES
2SC1815
2SC1959
GND
AUDIO


RES
BDØ-BD7
CAØ-CA15
CR/W
SØ2
UAB1
6522
UAB3
6522
UB4
7406
R17 1K
R18 1K
R19 1K
R16
10K
SERIAL PORT
SERIAL SRQ IN
SERIAL DATA OUT
SERIAL CLK IN
SERIAL DATA IN
SERIAL ATN OUT
SERIAL ATN IN
RESTORE
N.C.
KEYBOARD CONN
ROW Ø
ROW 1
ROW 2
ROW 3
ROW 4
ROW 5
ROW 6
ROW 7
COL Ø
COL 1
COL 2
COL 3
COL 4
COL 5
COL 6
COL 7
JOY 3
CASS MOTOR
JOY Ø
JOY 1
JOY 2
LITE PEN
CASS SWITCH
P7
POWER
SW1
F1 1A
L3 1.2µH
L4 1.5mH
CR2
DBA10B
LED
CN1
R23
390
FB1
FB2
Q3
2SC1815
Q4
2SD880
R20 1K
R21
10K
R22
1K
CR1
6.8V
C43
.01
CASS WRITE
CASS READ
CASSETTE P2
UC5
74LS138
BLK Ø
BLK 1
BLK 2
BLK 3
BLK 4
BLK 5
BLK 6
BLK 7
IRQ
NMI
UC2
74LS04
UC6
74LS138
COLOR
I/O2
I/O3
UB6
555
R24
220K
R25
150K
R26
1K
FB7
USER PORT P1
JP1
VR/W
CR/W
CDØ-CD7
RAM 1
RAM 2
RAM 3
AUDIO IN
P6
EXPANSION
Ⓒ Commodore Büromaschinen GmbH 1983
Jegliche Vervielfältigung oder Nachdruck
ohne Erlaubnis des Urheberrechtsinhabers
ist untersagt und wird auf dem
Rechtswege verfolgt.


U13
NC
U13
NC